**Advanced Cyber–Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things**
**Grant Agreement: 786698**

# D2.5 Threat actors' attack strategies

## Work Package 2: Cyber–threat landscape and end–user requirements

### Document Dissemination Level

| | | |
|---|---|---|
| PU | Public | X |
| CO | Confidential, only for members of the Consortium (including the Commission Services) | |

Document Due Date: 31/12/2018
Document Submission Date: 31/12/2018

**Co–funded by the Horizon 2020 Framework Programme of the European Union**

# Document Information

| | |
|---|---|
| **Deliverable number:** | D2.5 |
| **Deliverable title:** | Threat actors' attack strategies |
| **Deliverable version:** | 1.00 |
| **Work Package number:** | WP2 |
| **Work Package title:** | Cyber–threat landscape and end–user requirements |
| **Due Date of delivery:** | 31/12/2018 |
| **Actual date of delivery:** | 31/12/2018 |
| **Dissemination level:** | PU |
| **Editor(s):** | Konstantinos Limniotis (UOP) |
| **Contributor(s):** | Nicholas Kolokotronis, Costas Vassilakis, Nicholas Kalouptsidis, Konstantinos Limniotis, Konstantinos Ntemos, Christos–Minas Mathas, Konstantinos–Panagiotis Grammatikakis (UOP)<br>Dimitris Kavallieros, Giovana Bilali (KEMEA)<br>Stavros Shiaeles, Bogdan Ghita, Julian Ludlow, Salam Ketab, Hussam Mohammed, Abdulrahman Alruban (CSCAN) |
| **Reviewer(s):** | Pavué Clément (SCORECHAIN)<br>Michele Simioli (MATHEMA) |
| **Project name:** | Advanced Cyber–Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things |
| **Project Acronym** | Cyber–Trust |
| **Project starting date:** | 01/05/2018 |
| **Project duration:** | 36 months |
| **Rights:** | Cyber–Trust Consortium |

# Version History

| Version | Date | Beneficiary | Description |
|---|---|---|---|
| 0.10 | 22/10/2018 | UOP | Proposed deliverable’s outline |
| 0.15 | 06/11/2018 | UOP | Initial text in Section 5 has been written |
| 0.20 | 11/11/2018 | UOP | First draft of Section 4 |
| 0.25 | 30/11/2018 | UOP | First draft of Section 5 |
| 0.30 | 07/12/2018 | UOP | First draft of Section 3 |
| 0.35 | 09/12/2018 | UOP | New material added, presentation enhancements, and structural changes |
| 0.40 | 14/12/2018 | UOP | First draft of Section 1 |
| 0.45 | 15/12/2018 | CSCAN | First draft of Section 8 |
| 0.50 | 17/12/2018 | KEMEA | First draft of Section 7 |
| 0.55 | 18/12/2018 | UOP | First draft of Section 6 |
| 0.60 | 18/12/2018 | UOP | First draft of Section 2 |
| 0.65 | 19/12/2018 | UOP | Final draft of deliverable sent for review |
| 1.00 | 30/12/2018 | UOP | Accommodation of review comments and other minor corrections |

# Acronyms

| ACRONYM | EXPLANATION |
|---|---|
| ACT | Attack Countermeasure Tree |
| ADT | Attack Defense Tree |
| AFT | Attack Fault Tree |
| AG | Attack graph |
| AIV | Annual Infrastructure Value |
| ALE | Annual Loss Expectancy |
| API | Application Programming Interface |
| ARC | Annual Response Cost |
| ART | Attack response Tree |
| AT | Attack tree |
| BAG | Bayesian Attack Graph |
| CAG | Core Attack Graph |
| CMS | Content Management System |
| CoAG | Conservative Attack Graph |
| CPE | Common Platform Enumeration |
| CSV | Comma–Separated Values |
| CUI | Character User Interface |
| CVE | Common Vulnerabilities and Exposures |
| CVRF | Common Vulnerability Reporting Format |
| CVSS | Common Vulnerability Scoring System |
| CWE | Common Weakness Enumeration |
| DAG | Directed Acyclic Graph |
| DDoS | Distributed Denial of Service |
| DNS | Domain Name Server |
| DT | Defense Tree |
| EDG | Exploit Dependency Graph |
| eVDB | enriched Vulnerability Data Base |
| GCF | Greenbone Community Feed |
| GPL | General Public License |
| GPO | Group Policy Object |
| GPRS | General Packet Radio Service |
| GPS | Global Positioning System |
| GrSM | Graphical Security Model |
| GSF | Greenbone Security Feed |
| HARM | Hierarchical Attack Representation Model |
| HTTP | Hypertext Transfer Protocol |
| HVAC | Heating, Ventilation, and Air Conditioning |
| IDPS | Intrusion Detection and Prevention System |
| IDS | Intrusion Detection System |
| IEC | International Electrotechnical Commission |
| iIRS | intelligent Intrusion Response System |

| | |
|---|---|
| **IoT** | Internet of Things |
| **IPS** | Intrusion Prevention System |
| **ISO** | International Standards Organization |
| **LGA** | Logical Attack Graph |
| **NASL** | Nessus Attack Scripting Language |
| **NCCIC** | National Cybersecurity and Communications Integration Center |
| **NFC** | Near Field Communication |
| **NGFW** | Next Generation FireWall |
| **NIST** | National Institute of Standards and Technology |
| **NSE** | Nmap Scripting Engine |
| **NVD** | National Vulnerability Database |
| **OS** | Operating System |
| **OSINT** | Open–Source INTelligence |
| **OVAL** | Open Vulnerability and Assessment Management |
| **OWAT** | Ordered Weighted Averaging Tree |
| **PAG** | Personalized Attack Graph |
| **PCAP** | Packet Capture |
| **PT** | Protection Tree |
| **RDF** | Resource Description Framework |
| **RM** | Risk Mitigation |
| **SCAP** | Security Content Automation Protocol |
| **SCT** | Security Compliance Toolkit |
| **SDN** | Software Defined Network |
| **SNMP** | Simple Network Management Protocol |
| **SQL** | Structured Query Language |
| **TCP** | Transmission Control Protocol |
| **TMS** | Trust Management Service |
| **TVA** | Topological Vulnerability Analysis |
| **UDP** | User Datagram Protocol |
| **UI** | User Interface |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **VDB** | Vulnerability Data Base |
| **VM** | Virtual Machine |
| **VPN** | Virtual Private Network |
| **XML** | eXtensible Markup Language |

# Table of Contents

# List of Figures

# List of Tables

# Executive summary

This report focuses on providing the necessary information and methodologies for modelling the possible attack strategies used by threat actors of particular profiles in selected types of cyber–attacks targeting at devices, networks and *critical information infrastructures* (CIIs). Since the ecosystem of IoT devices is highly heterogenous, based on devices with different characteristics and processing operations, a systematic approach to model attack strategies of several forms – taking also into account the various capabilities of the potential attackers – becomes prerequisite for the process of adopting and evaluating the proper mitigation measures with respect to the relevant risks.

Towards efficiently modelling the attack strategies, there exist numerous applications that can be used to acquire the necessary information, whilst there are also several risk management approaches. Moreover, the so–called Graphical Security Models constitute important primitives for efficiently representing the various attack strategies; they are based on input information (i.e. software weaknesses, misconfigurations, network connectivity etc.) to identify – via employing appropriate algorithms – the possible attack steps that can be executed, as well as the relevant consequence. Appropriate Graphical Security Models may also allow for developing a systematic risk management, thus resulting in appropriate mitigation measures.

The present deliverable surveys all the available tools and methodologies for a concrete modelling of attack strategies, performing a comparative study in terms of well–defined criteria. These tools and methodologies include: i) tools for information acquisition (network topology, host connectivity, vulnerabilities), ii) description of graphical security models, iii) methods for feeding these models with the information obtained, iv) tools and algorithms for building and utilizing such models, v) risk management approaches, as well as vi) tools for enforcing mitigation. By these means, a systematic approach to efficiently model the possible attack strategies towards adopting appropriate defensive actions in relation with the likelihood of the attacks is being constructed. Practical realistic examples in the framework of the Cyber–Trust use cases are also described, whilst relevant simulation environments are also discussed. The output of this deliverable, based on the aforementioned analysis, is the description of the relevant approach that will be followed in the framework of the Cyber–Trust project.

The deliverable provides a thorough analysis of tools and methods for the efficient modelling of attacker’s strategies in the context of Cyber–Trust; it is therefore quite technical by nature. We believe that readers with a technical background will find the presentation quite comprehensive and the analysis accurate and complete. Non-technical readers might have to skip more technical parts, especially during the first reading.

# 1. Introduction

The Cyber–Trust project aims to develop an innovative cyber–threat intelligence gathering, detection, and mitigation platform to tackle the grand challenges towards securing the ecosystem of IoT devices. These challenges rest with the special structure of IoT networks, that is heterogeneous connected devices – computers, laptops, smartphones, and tablets, as well as, embedded devices and sensors – communicate via exchanging large volumes of data. For example, security issues occur from embedded devices and other legacy hardware, whose flawed design or their poor configuration allows the cyber–criminals to compromise them in order to mount a successful attack. Therefore, it is of high importance to quickly detect, effectively respond to and mitigate sophisticated cyber–attacks. To this goal, *a systematic approach* to model several attack strategies becomes essential, so as to properly identify the possible weaknesses of the system, the relevant risks in relation with the probability of an attack being successful, as well as the effective measures that need to be taken towards addressing these security issues, both proactively and reactively; this is a non–trivial task, taking into account the inherent complexity of the system, as well as the fact that new vulnerabilities – and, thus, relevant risks – are constantly arising.

## 1.1 Purpose of the document

This document aims to provide a modelling of attackers and attack strategies with respect to potential cyber–attacks targeting at any part of the Cyber–Trust platform (devices, networks and CIIs). Such a modelling will in turn allow for developing appropriate mitigation measures, being either proactive or reactive.

More precisely, a proper identification of attackers' profiles is essential in effectively addressing the security threats, as well as in appropriately responding to cyber–attacks. Constructing attackers' profiles rests with considering the attacker as an entity with varying (depending on the profile) constrained resources, like budget, tools, etc., aiming at exploiting vulnerabilities of any kind to maximize his profit (access level, degrade QoS, etc.). Depending on the profile, some attack strategies will be more probable than others. Therefore, the attack strategies should be modelled in a systematic way to confront them. To this end, there are known tools to model the attack strategies – the most prominent being attack trees and attack graphs. These tools allow for presenting the possible paths that an attacker of any kind might follow (possibly in an adaptive manner) towards achieving his goals, whilst they also provide information on what needs to be done to alleviate security issues.

Utilizing appropriate tools to model attack strategies necessitates collection of appropriate information, including information on the network topology, on reachability amongst several nodes/devices (e.g. information on firewall rules), as well as on devices/software vulnerabilities (which in turn is contingent on system's elements configuration). All these pieces of information should somehow feed the attack model, which will be developed in terms of appropriately estimating and combining the so–called preconditions that must be met with respect to exploiting specific vulnerabilities, as well as the so–called postconditions corresponding to the consequences occurred in case that some attackers' actions succeed. Moreover, such modeling tools for attack strategies allow for performing a risk analysis on the overall system, taking into account the relevant vulnerabilities and their corresponding probabilities of occurrence in conjunction with their impact. These systematic procedures allow for properly identifying and evaluating possible weaknesses, which in turn result in making proper decisions with regard to the security measures (mitigation steps) that need to be implemented.

This document presents an overview of the available methods to model attack strategies, whilst it also surveys the suitable software tools of any type that can be used within the framework of the Cyber–Trust platform towards implementing such methods. In this context, typical scenarios of potential attack strategies in the Cyber–Trust use cases are also given. The ultimate goal is to define a specific approach that will be adopted in the case of the Cyber–Trust platform, taking into account its special characteristics and requirements.

## 1.2 Relations to other activities in the project

The computation of the cyber–attack security model, quantifying the impact of the corresponding mitigation actions, is an essential building block towards achieving security of the overall Cyber–Trust platform; this is also reflected in the Cyber–Trust use case scenarios in D2.3. To this end, this document will provide useful input to task T5.3 that focuses on building an autonomous cyber–defense framework to cope with intelligent cyber–attackers, as well as to tasks T6.2 and T6.3 which rest with developing techniques for detecting and mitigating attacks. It should be also pointed out that, in practice, computing the cyber–attack security model is strongly related with the cyber–threat landscape, which has been reviewed and analyzed in T2.1.

## 1.3 Structure of the document

This document consists of nine sections, including the current introductory section. More precisely, the structure of the document is as follows:

- Section 2 briefly describes the overall methodology adopted in the present document towards deriving the specific approaches that will be adopted in the process of efficiently modelling the attackers and attack strategies with respect to potential cyber–attacks targeting at the Cyber–Trust platform.
- Section 3 analyzes – in terms of evaluating their specific characteristics via a comparative study – the available methods and tools to handle the appropriate information acquisition and exploitation within the Cyber–Trust platform, so as to subsequently perform risk–based cyber threat mitigation.
- Section 4 provides an overview and a comparative study of the so–called *Graphical Security Models* (GrSMs), which constitute a powerful tool for carrying out a systematic analysis of security weaknesses of systems and evaluating potential protection measures against cyber–attackers.
- Section 5 describes the known software tools for exploiting *attack graphs* (which seem to be the most prominent GrSM), in conjunction with their underlying algorithms, whilst a discussion on their applicability in the context of the Cyber–Trust is also provided.
- Section 6 provides a comprehensive review on risk management and attack mitigation approaches, focusing on the information systems level so as to address the needs of Cyber–Trust project. A detailed study of the available mitigation tools is also given.
- Section 7 focuses on a classification of the attackers, describing for each case their relevant available resources as well as the skills needed towards mounting cyber–attacks.
- Section 8 presents specific examples of attack strategies in the context of the Cyber–Trust use–cases, to illustrate the importance and applicability of the previously described methodologies. These examples are based on typical (realistic) network setups in the domains of interest, taking also into account specific (potential) characteristics on the devices, OS and services, versions, etc. A discussion on relevant simulation environments is also given.
- Finally, the main conclusions obtained are summarized in Section 9.

## 2. Methodology

Securing a network from advanced cyber–attacks is a primary concern for IT security officers. Such attacks have become more frequent and even more sophisticated due to the vast number of networked devices and the security problems arising from embedded and legacy hardware. Even though the critical assets in a large corporate or a *small–office, home–office* (SOHO) network (the latter being among the project's application domains) are protected by firewalls, vulnerabilities that exist in other devices (from which the critical assets are reachable) can be used as pivot to launch multi–stage correlated attacks. This was already highlighted in the description of task T2.4, where the need was identified for modeling cyber–attackers' strategies via a *graphical security model* (GrSM) like attack trees and attack graphs.

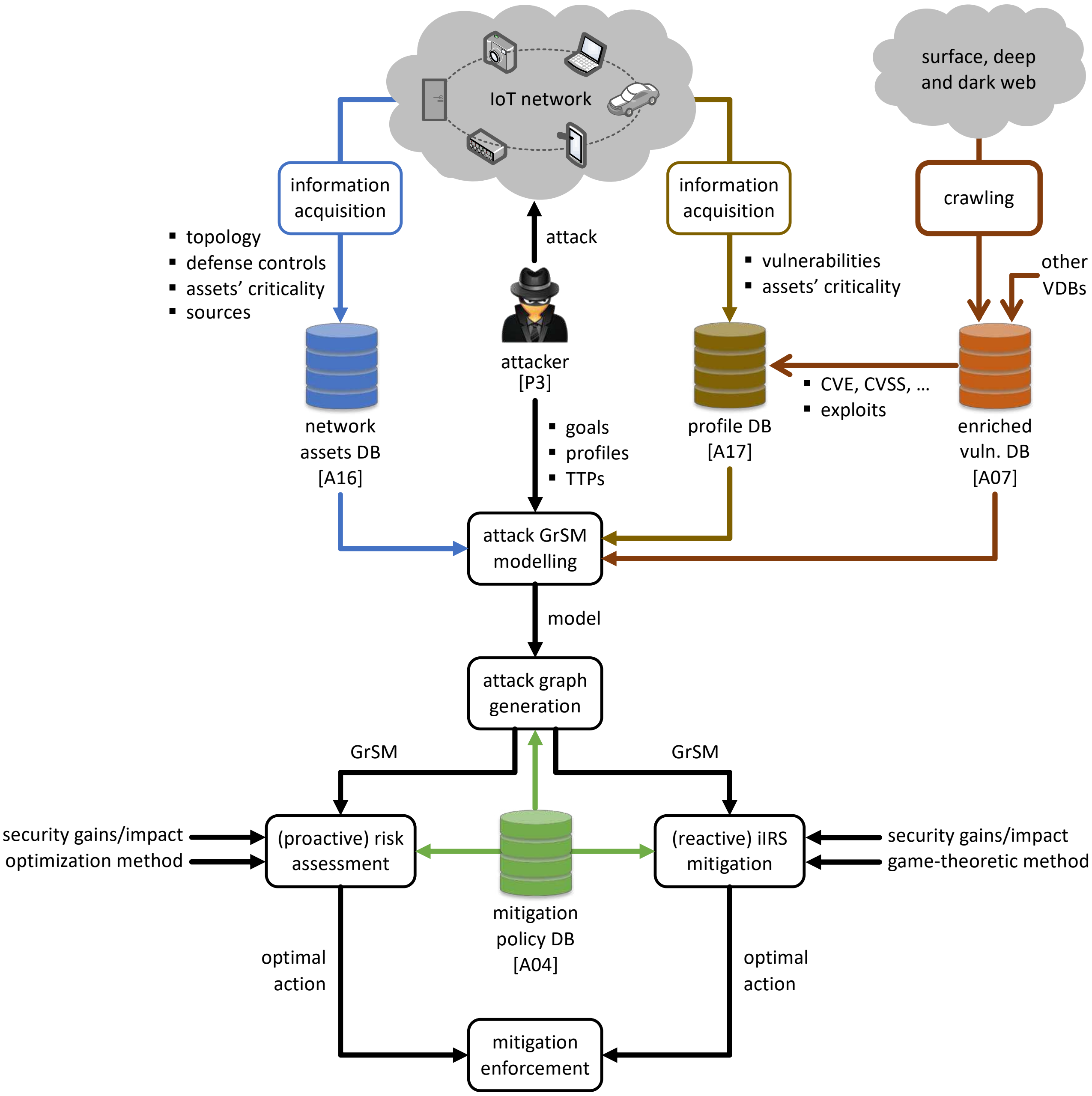


Figure 2.1. Generation of threat actors' attack strategies and application in the mitigation process

Many tools to detect open ports and vulnerabilities on a network's devices exist but their limitation is that they identify the vulnerabilities *on a per host basis* and thus they are *unable* to detect sophisticated correlated attacks that usually occur in complex and dynamic environments as in the case of IoT. In addition, not many automated penetration testing tools, which are employed in typical network security analysis, are available for the active security/risk assessment of devices. Therefore, in order to get a holistic view of a network's health status, the security officers need to take these correlated attacks into consideration. The approach considered in Cyber–Trust, i.e., to assess security via GrSMs requires a number of steps that are illustrated in Figure 2.1 (where person–class and assets–class actors presented in deliverable D2.3 have been included – see also Table 2.1 for a mapping of the illustrated processes to the project's asset–class actors); this involves the consideration of the following aspects:

- acquiring information about a network and devices' vulnerabilities;
- modeling the different attack scenarios/paths that attackers' might follow; and
- recommending mitigation actions in an intelligent way.

Table 2.1. Mapping of processes to Cyber–Trust's asset–class actors

| Processes | Responsible asset–class actors |
|---|---|
| Information acquisition | [A03] Monitoring service<br>[A16] Profiling service |
| Attack GrSM modeling & Attack graph generation | [A05] Trust management system<br>[A13] Smart gateway iIRS application<br>[A14] Smart device iIRS application |
| Risk assessment | [A05] Trust management system |
| iIRS mitigation | [A13] Smart gateway iIRS application<br>[A14] Smart device iIRS application |
| Mitigation enforcement | [A04] Cyber-defense service |
| Crawling | [A10] Crawling service |

Our goal is to leverage open–source scanning tools that gather vulnerability and other network information, by using the project's *enriched vulnerability database* (eVDB), and use them to generate attack graphs for delivering advanced security assessment and intelligent mitigation strategies by relying on the *intelligent intrusion response system* (iIRS). The mitigation actions that will be output from the iIRS will consider not only the attack graph's properties, but also the security gains and impacts that an action will have (in the long–term) on a network's security, operation, etc., and the predefined mitigation policies of organizations. The block diagram illustrated in Figure 2.1 utilizes the following asset–class actors of Cyber–Trust (*see* D2.3):

- *Information acquisition:* refers to the collection of information related to a network that is stored in the network architecture and assets repository [A16] and the profile database [A17].

  Common methodologies probing hosts for open ports, identifying running services/applications, and performing vulnerability assessment (i.e., determining, quantifying, and ranking vulnerabilities) will be used to obtain information about list of hosts and services, host–to–host connectivity, sources of attack, and goals of attack. Vulnerabilities will be correlated with eVDB [A07] to get an extended set of information.

- *Attack GrSM modeling:* takes as input all the information collected in the previous step in order to represent the cyber–attacks (realized by exploits) in a machine–readable form.

This step is carried out by a module shared by the *trust management system* (TMS) [A05] and the iIRS [A13, A14] since it is subsequently used for dynamic risk assessment and intelligent mitigation respectively. The available exploits are commonly modeled as a set of preconditions (necessary for triggering an exploit) and postconditions (the effects of an exploit's execution).

- *Attack graph generation:* this is where the actual attack graph is generated by using specialized and efficient algorithms able to determine all possible attack paths that an attacker might follow in order to achieve his goals.
- *Mitigation actions computation:* such actions are computed in a proactive manner (by conducting risk assessment) and reactive manner (via the iIRS) by considering the usual trade–off between the level of security enhancements (gains) and the cost of mitigation (impact).

The output of this work will be the concrete process (and the tools to be used) for modeling attack strategies that attackers of a profile might follow in the course of a cyber–attack as a response to defensive actions taken. This deliverable will provide input to tasks T5.2–3 and T6.2–3 to help defining the defensive actions of the IoT devices. In the subsequent sections, we describe these steps in more detail.

# 3. Information acquisition

This section discusses the methods and tooling that are available to handle the different phases of information acquisition and exploitation within the Cyber–Trust platform, so as to perform risk–based cyber threat mitigation. More specifically:

1. The Cyber–Trust platform needs to have available the list of devices that are within its domain of protection. To this end, tooling for identifying active devices within this domain is required. Besides the plain identification of individual hosts and their addresses, a multitude of additional information can be exploited to better assess vulnerability and threat levels, including network topology (including subnets and device–to–subnet mapping), operating systems running on the devices and their versions, host reachability, services running on the devices and their versions etc. Furthermore, the risk level that a device is exposed to due to the existence of some threat is clearly dependent on the potential of the threat agent to reach the vulnerabilities of the device. Taking into account that routing rules or defense measures may preclude packets originating from a specific host to reach a specific device or service, it is evident that host and service reachability is an additional piece of information that must be collected. Section 3.1 reviews the available tools and their features.

2. Threat agents seek to exploit device vulnerabilities, to achieve breaches. Therefore, the Cyber–Trust platform needs to maintain a comprehensive list of the vulnerabilities applicable to each device it protects, in order to both take automated actions to disrupt attempts exploiting these vulnerabilities and also raise appropriate awareness events for the device owners and Cyber–Trust platform operators. The available tools for vulnerability scanning are examined in Section 3.2, and their suitability for the context of the Cyber–Trust platform is assessed through a number of criteria.

3. In order for an exploit targeting a specific vulnerability to succeed, certain *preconditions* must typically be met. This extends beyond simple network connectivity to the target device or reachability to the target service, and may include aspects such as holding some level of privileges (to achieve privilege elevation/escalation) or even knowledge (on the attacker side) regarding the services that run on a device, their version and configuration. The conjugate to preconditions regarding attacks are the *postconditions*, which correspond to the consequences inflicted when some attack succeeds. When preconditions and postconditions are known, attacker strategies may be modeled using graphical security models (see Section 4), which can be used to predict an attacker's behavior and effectively select and apply the proper defense and mitigation measures. However, insofar, the identification of the pre– and post–conditions is a key bottleneck in the usability and effectiveness of these graphical security models. Considering these aspects, ways to acquire intelligence regarding exploits, particularly focusing on preconditions and postconditions are examined in Section 3.3.

4. Finally, the Cyber–Trust platform should be able to mitigate threats, both via proactive and reactive measures. Information about the prominent reaction methods may be present in various sources, including vendor product and patch pages, vulnerability databases etc. This information is typically listed in free text, in non–standardized formats, and in varying levels of detail, thus, encumbering the use of automated methods for its identification, extraction and use. Moreover, the application of certain measures –especially reactive ones– may have effects on the value of the protected assets, e.g., demote availability while limiting access to a service to guard against more grave effects. The means to mitigate attacks and the issues to be tackled in this process are presented in Section 3.4.

## 3.1 Network topology and host connectivity

In this section a number of tools will be presented and compared based on a list of characteristics; these include both functional capabilities related to gathering information about a target network and non–functional ones, such as the license. The features are presented in Table 3.1 below.

Table 3.1. Features against which network topology and host connectivity tools are compared

| Feature | Possible values | Description |
|---|---|---|
| Active hosts | X/– | Identification of hosts that are active within the scanned networks. |
| Reachability | X/– | Identification of hosts/services that are reachable within the scanned networks. |
| Network topology | X/– | Extraction of network topology elements, focused on segmentation of the network in subnets, presence of interconnecting routerns and host membership in identified subnets. |
| Existence of UI and/or visualization capabilities | Textual description | Description of the ways that the tool presents information to the user and generally interfaces with users; command line and graphical UIs are examined, as well as visualization capabilities. |
| Output formats | Textual description | Different ways that output formats can be stored (e.g. CSV, XML) are examined. |
| OS and version | X/– | Whether the tool can determine the OS that enumerated hosts run, as well as their versions. |
| Active ports | X/– | Whether the tool can identify the ports that are open in enumerated hosts. |
| Services and versions | X/– | Whether the tool can determine the services listening to the open ports, as well as their versions. Note that this goes beyond simple lookup of port numbers in lists of well–known service port assignments[1]; here we consider lookup of service or protocol signatures within the data returned by the service in response to suitably crafted requests. |
| Analysis of log files vs. active scanning | Textual description | This feature pertains to whether the tool needs to actively monitor and analyze network traffic, or whether it can read and process traffic data captured in respective files (typically *pcap–type* files, but other file types can be used) resulting thus in an offline analysis scheme. |
| License | Textual description | The license under which the software is made available; this includes fees/price, the ability to create derivatives and the license scheme that derivatives should/can be made available. |

*The marks 'X' and '–' correspond to Yes and No respectively; if such information is not available, this is noted with '?'.*

In addition to the above, a number of reconnaissance tools is presented, and the features of interest are illustrated in Table 3.2.

Table 3.2. Features against which reconnaissance tools are compared

| Feature | Possible values | Description |
|---|---|---|

[1] https://www.iana.org/assignments/service–names–port–numbers/service–names–port–numbers.xhtml

| | | |
|---|---|---|
| Domain and subdomain names | X/– | The capability of the tool to gather domain and subdomain names associated with scan target. |
| IP addresses | X/– | The capability of the tool to gather a list of IP addresses associated with scan target. |
| Virtual hosts | X/– | The capability of the tool to identify virtual hosts running on web servers of the scan target. |
| Open ports, services, banners | X/– | Whether the tool is able to scan the target network for open ports, identify the services listening to those ports and analyze banners presented by the services. |
| Target spec | Textual description | The list of information items that the tool is able to gather. |
| License | Textual description | The license under which the software is made available; this includes fees/price, the ability to create derivatives and the license scheme that derivatives should/can be made available. |
| E–mail addresses and peoples' names | X/– | Whether the tool is able to gather e–mail addresses and names of persons associated with the scan target. |
| OS and version | X/– | Whether the tool can determine the OS that enumerated hosts run, as well as their versions. |
| Applications and their components | X/– | Whether the tool can identify the applications used in the scan target and their components. |
| UI types | Desktop/<br>Command line/<br>Web–based | Description of the ways that the tool presents information to the user and generally interfaces with users; command line, desktop and web–based UIs are examined. |
| Output options | Textual description | Different ways that output formats can be stored (e.g. CSV, XML) are examined. |

*The marks 'X' and '–' correspond to Yes and No respectively; if such information is not available, this is noted with '?'.*

### 3.1.1 List of tools considered

In the following sections a non–exhaustive list of eighteen available (known) tools is presented; these are the network topology and host connectivity tools Nmap, Angry IP scanner, Unicornscan, Dipiscan, Masscan, Scanrand, Zmap, NetCrunch tools, MyNet toolset, LanTopoLog, Spiceworks network mapping, Network-Miner, PcapViz, and Skydive, along with the reconnaissance tools Maltego, Netglub, and Dnsdumpster.com.

#### *3.1.1.1 Nmap*

Nmap[2], abbreviation of network mapper, is an open–source software for network discovery and security testing. It is widely used from network administrators and penetration testers, but also from malicious users. Its most common usage is port scanning; however, it has a lot more to offer than that.

*Nmap* sends specially crafted packets in order to determine which devices are active on the network, the services and their version running on these devices, their operating system and what kind of security measures are deployed in the network (IP/packets filtering, firewalls, etc.). Furthermore, *nmap*'s capabilities are extended by the usage of the NSE (Nmap Scripting Engine), which is a collection of scripts for vulnerability scanning, default credentials detection, advanced service detection and many more. All of the above are

[2] https://nmap.org/

supported by a large community and updated regularly. NSE allows integration of custom–made scripts written using the LUA language in the *nmap* functionality and can be plugged into the processes of network discovery (to provide more information about existing network elements), version detection (for more elaborate version identification), vulnerability detection (leveraging the basic capabilities bundled into nmap) and backdoor detection (for more sophisticated detection of backdoors). NSE can be also used to perform vulnerability exploitation, a feature typically used in penetration testing, although not envisioned to be used in the context of Cyber–Trust.

Nmap was initially designed for Linux operating systems, but now it is available for many popular operating systems including Windows and Mac OS X. As mentioned above, *nmap* can also perform vulnerability scanning. For more information the user is referred to Section 3.2.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | X (both built–in and nmapscan.pl[3]) |
| **Existence of UI and/or visualization capabilities** | Zenmap[4]; nmapscan.pl also includes visualization capabilities; fe3d[5] visualizes network structures collected by nmap |
| **Output formats** | Redirection of standard output, XML, Grepable, Script kiddie |

| | |
|---|---|
| **OS and version** | X |
| **Active ports** | X |
| **Services and versions** | X |
| **Analysis of log files vs. active scanning** | Active scanning, Analysis of log files (Zenmap) |
| **License** | GPL v2 |

### *3.1.1.2 Angry IP scanner*

Angry IP Scanner[6] is a widely used open–source and multi–platform network scanner. It is extensible through plugins and very user–friendly. It is used by network administrators, penetration testers and so on. Its capabilities include but are not limited to port scanning, active host discovery, host and domain name detection and services/version detection. Furthermore, anyone with Java coding knowledge can extend its functionality by writing plugins. Additionally, Angry IP Scanner offers various output formats. Finally, it is considered to be really fast because of its multi–threaded approach, where a separate scanning thread is created for each scanned IP address.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | X |
| **Existence of UI and/or visualization capabilities** | X (Desktop UI) |

| | |
|---|---|
| **OS and version** | X |
| **Active ports** | X |
| **Services and versions** | X |
| **Analysis of log files vs. active scanning** | Active scanning |

[3] https://github.com/tedsluis/nmap/blob/master/nmapscan.pl
[4] https://nmap.org/zenmap/
[5] https://sourceforge.net/projects/fe3d/
[6] https://angryip.org/

| | | | |
|---|---|---|---|
| **Output formats** | CSV, TXT, XML, IP–Port list | **License** | GPL v2 |

*3.1.1.3 Unicornscan*

Unicornscan[7] is an information gathering and correlation engine built for and by members of the security research and testing communities. It is an attempt at a User–land Distributed TCP/IP stack. Some abilities include, asynchronous stateless TCP scanning/banner grabbing, asynchronous protocol–specific UDP scanning and active and passive remote OS, application, and component identification by analyzing responses. Additional functionalities include *pcap* file logging and filtering, relational database output, custom module support and customized data–set views. It is available for Linux, BSD, Solaris and Mac OS X.

| | | | |
|---|---|---|---|
| **Active hosts** | X | **OS and version** | X |
| **Reachability** | – | **Active ports** | X |
| **Network topology** | – | **Services and versions** | X |
| **Existence of UI and/or visualization capabilities** | – | **Analysis of log files vs. active scanning** | Active scanning |
| **Output formats** | Stdout redirection to log file, relational database, pcap file with received packets | **License** | GPL v2 |

*3.1.1.4 Dipiscan*

Dipiscan[8] is a portable network scanner for Windows devices to run scans on their local area network to detect network devices. For every device detected some information is given if available, some of the information returned includes, NetBIOS name, DNS name, Domain, and OS. It has the ability to scan by IP range, NetBIOS name and DNS name. Additionally, it provides a trace route functionality.

| | | | |
|---|---|---|---|
| **Active hosts** | X | **OS and version** | X |
| **Reachability** | X | **Active ports** | – |
| **Network topology** | X | **Services and versions** | X (NetBIOS only, when user rights permit so, through the Windows service management console) |
| **Existence of UI and/or visualization capabilities** | X | **Analysis of log files vs. active scanning** | Active scanning |
| **Output formats** | TXT | **License** | Freeware |

[7] https://tools.kali.org/information–gathering/unicornscan/

[8] https://www.dipisoft.com/

#### *3.1.1.5 Masscan*

Masscan[9] is a port scanner and is considered to be the fastest one. Its regular output is similar to that of nmap, but internally it uses asynchronous transmission. It also uses a custom TCP/IP stack.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | – |
| **Network topology** | – |
| **Existence of UI and/or visualization capabilities** | – |
| **Output formats** | XML, binary, grepable, JSON, list |

| | |
|---|---|
| **OS and version** | – |
| **Active ports** | X |
| **Services and versions** | X |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | A–GPL–3 |

#### *3.1.1.6 Scanrand*

Scanrand[10] is a network scanning tool designed to scan large networks very fast. It creates two completely separate and disconnected processes; one that sends queries and one that receives responses and reconstructs the original message from the returned content. Additionally, the receiving process doesn't retain state, it works by using a stateful protocol (TCP) in a stateless way.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | X (distance from scanning host) |
| **Existence of UI and/or visualization capabilities** | – |
| **Output formats** | Dump in SQL database |

| | |
|---|---|
| **OS and version** | – |
| **Active ports** | X |
| **Services and versions** | – |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | BSD original |

#### *3.1.1.7 Zmap*

Zmap[11] is an open–source network scanner developed as a faster alternative to nmap. It can conduct Internet–wide network surveys efficiently, more specifically it is claimed to be able to scan the entire IPv4 address space in under 45 minutes. Zmap uses what is called cyclic multiplicative groups, which allows it to scan roughly 1,300 times faster than nmap. However, its functionality is limited as compared to nmap; external applications can be used to supplement it.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | – |

| | |
|---|---|
| **OS and version** | – |
| **Active ports** | X (typically a single port is scanned; invoking the tool multiple times can be |

[9] https://github.com/robertdavidgraham/masscan/

[10] https://www.darknet.org.uk/2007/12/scanrand–download–stateless–tcp–scanner–with–syn–cookies/

[11] https://github.com/zmap/

| | |
|---|---|
| | |
| **Network topology** | – |
| **Existence of UI and/or visualization capabilities** | – |
| **Output formats** | Stdout redirection, CSV, Redis, JSON |

| | |
|---|---|
| | used to enumerate ports) |
| **Services and versions** | X (via external banner grabbing applications) |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | Apache license 2 |

### *3.1.1.8 NetCrunch tools*

NetCrunch tools[12] is a free network tools collection which provides a UI and runs on Windows. It provides three categories of tools, the *basic IP tools* which include tools like Traceroute and DNS Info, the *subnet tools* which include tools like MAC resolver and Subnet Calculator and the *scanners* which include tools like network service scanner and open TCP port scanner. It doesn't offer any export options. The program is free to use but requires a registration.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | – |
| **Existence of UI and/or visualization capabilities** | X |
| **Output formats** | – |

| | |
|---|---|
| **OS and version** | – |
| **Active ports** | X |
| **Services and versions** | X |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | adremsoft.com/netcrunch.tools/eula/ |

### *3.1.1.9 MyNet toolset*

MyNet toolset[13] is a free network mapping toolset provided by AdRem. It detects all the network nodes connected to the local network and displays them in a graph. It also scans each node for popular services that might be running. It provides more details for each node including name, DNS, IP and MAC address. For each node there is an option to access a set of network tools: ping, traceroute, and more. It runs on Windows.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | – |
| **Existence of UI and/or visualization capabilities** | X |
| **Output formats** | – |

| | |
|---|---|
| **OS and version** | – |
| **Active ports** | – |
| **Services and versions** | X |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | Freeware |

[12] https://www.adremsoft.com/

[13] https://www.adremsoft.com/mynettoolset/

### *3.1.1.10 LanTopoLog*

LanTopoLog[14] is an application that provides physical network topology discovery based on SNMP, visualization and monitoring. It provides many functionalities including detection of new devices and notification of the event, real–time device status monitoring, web browser–based access from anywhere in the network and visualization of the topology. Runs on Windows.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | X |
| **Existence of UI and/or visualization capabilities** | X |
| **Output formats** | CSV |

| | |
|---|---|
| **OS and version** | X |
| **Active ports** | – |
| **Services and versions** | – |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | Shareware; the free version disables some features after a specific period of time. |

### *3.1.1.11 Spiceworks NM*

Spiceworks NM (network mapping)[15] is a network mapping and management software. It provides a graphical interface where a complete and customizable map of the network is presented. Some of its features include analyzation of the bandwidth usage between the nodes, device details and network problems diagnostics. Runs on Windows.

| | |
|---|---|
| **Active hosts** | X |
| **Reachability** | X |
| **Network topology** | X |
| **Existence of UI and/or visualization capabilities** | X (browser based) |
| **Output formats** | A number of reports is available, which can be saved in CSV, XLS and PDF |

| | |
|---|---|
| **OS and version** | X |
| **Active ports** | X |
| **Services and versions** | X |
| **Analysis of log files vs. active scanning** | Active scanning |
| **License** | Free after registration |

### *3.1.1.12 NetworkMiner*

NetworkMiner[16] is an open–source network forensic analysis Tool that runs on Windows, Linux, Mac OS X and comes in free and professional editions. It is able to detect operating systems, sessions, hostnames, open ports etc. by using passive network sniffing and packet capturing without putting any traffic on the network. It can also perform offline analysis with *packet capture* (pcap) files as input.

| | |
|---|---|
| **Active hosts** | X |

| | |
|---|---|
| **OS and version** | X |

[14] https://www.lantopolog.com/

[15] https://www.spiceworks.com/free–network–mapping–software/

[16] https://www.netresec.com/?page=Networkminer

| | |
|---|---|
| Reachability | – |
| Network topology | – |
| Existence of UI and/or visualization capabilities | X |
| Output formats | Export to CSV / Excel / XML / CASE / JSON–LD (paid version only) |

| | |
|---|---|
| Active ports | X |
| Services and versions | X |
| Analysis of log files vs. active scanning | Analysis of pcap files and passive scanning |
| License | GPLv2; paid version option |

### *3.1.1.13 PcapViz*

PcapViz[17] visualizes network topologies and provides graph statistics based on pcap files. Makes the determination of key topological nodes and data exfiltration attempts easier. Amongst others, its features include: (a) drawing of network topologies (Layer 2) and communication graphs (Layer 3 and 4); (b) inclusion of country information and connection stats in network topologies; and (c) collection of statistics, such as most frequently contacted machines.

| | |
|---|---|
| Active hosts | – |
| Reachability | X |
| Network topology | X |
| Existence of UI and/or visualization capabilities | X (GraphViz, dot) |
| Output formats | output redirection |

| | |
|---|---|
| OS and version | – |
| Active ports | – |
| Services and versions | – |
| Analysis of log files vs. active scanning | Analysis of pcap files |
| License | N/A |

### *3.1.1.14 Skydive*

Skydive[18] is an open source real–time network topology and protocols analyzer that collects, stores and analyzes the state of network infrastructure and the flows going through this infrastructure. Furthermore, Skydive is SDN–agnostic, which means it doesn't rely on SDN solutions but provides a way to collect information from SDN controllers. Its core features include:

- Capture of network topology and flows.
- Full history of network topology and flows.
- Distributed architecture.
- Support for VMs and containers infrastructure.
- Unified query language for topology and flows (Gremlin).
- REST API.

Skydive is composed of two components, namely the Skydive Agent and the Skydive Analyzer. The Skydive agents collect topology information and flows and forward them to a central agent for further analysis. All the information is stored in an Elasticsearch database.

| | |
|---|---|
| Active hosts | – |
| Reachability | X |

| | |
|---|---|
| OS and version | – |
| Active ports | – |

[17] https://github.com/mateuszk87/PcapViz
[18] http://skydive.network/

| | |
|---|---|
| **Network topology** | X |
| **Existence of UI and/or visualization capabilities** | X |
| **Output formats** | All facilities provided by Kibana and other Elastic search clients |

| | |
|---|---|
| **Services and versions** | – |
| **Analysis of log files vs. active scanning** | Collection and analysis of log files |
| **License** | Apache 2.0 |

### *3.1.1.15 Maltego*

Maltego[19] is a network reconnaissance and data mining tool that gathers information from open sources and visualizes it in a graph. It can analyze relationships between information that is publicly accessible on the Internet, e.g. footprinting Internet infrastructure and finding information people and organizations. The connections are found using OSINT by querying sources such as DNS records, whois records and social networks. Additionally, it can import/export the graph result in many formats (CSV, XLS, PDF, image formats). It is available in both free and paid versions.

| | |
|---|---|
| **Domain and subdomain names** | X |
| **IP addresses** | X |
| **Virtual hosts** | X |
| **Open ports, services, banners** | X |
| **Target spec** | Domain, DNS Name, IPV4 Address, MX Record, NS Record, Autonomous System (AS), etc. |
| **License** | Community and paid editions |

| | |
|---|---|
| **E–mail addresses and peoples' names** | X |
| **OS and version** | X |
| **Applications and their components** | X |
| **UI types** | Desktop |
| **Output options** | CSV, XLS, XLSX, PDF, image formats, GraphML, Entity Lists |
| | |

### *3.1.1.16 Netglub*

Netglub[20] is an open–source data information gathering and data mining tool that presents the information gathered in a graph that is easily understood. Practically, it's the open–source alternative to Maltego, but has limited documentation. It doesn't have sufficient documentation, it isn't maintained properly and has less functionality than Maltego and it is less user–friendly.

| | |
|---|---|
| **Domain and subdomain names** | X |
| **IP addresses** | X |

| | |
|---|---|
| **E–mail addresses and peoples' names** | X |
| **OS and version** | – |

[19] https://www.paterva.com/web7/

[20] http://www.netglub.org/

| | | | |
|---|---|---|---|
| **Virtual hosts** | X | **Applications and their components** | – |
| **Open ports, services, banners** | – | **UI types** | Desktop |
| **Target spec** | Domain, DNS name, IP address, IP subnetwork, URL, website, MX record, NS record, email address, person, phrase | **Output options** | CSV |
| **License** | GPL v3 | | |

### 3.1.1.17 *Dnsdumpster.com*

DNSdumpster.com[21] is a free domain research web application that can discover hosts related to a domain. It is able through DNS lookup and crawling to find extensive information related to a domain. It doesn't document all its capabilities, so the following table has been filled based on tests that we have performed.

| | | | |
|---|---|---|---|
| **Domain and subdomain names** | X | **E–mail addresses and peoples' names** | – |
| **IP addresses** | X | **OS and version** | X |
| **Virtual hosts** | – | **Applications and their components** | X |
| **Open ports, services, banners** | X | **UI types** | Web–based |
| **Target spec** | Domain | **Output options** | XLSX |
| **License** | Free, with limitations on the number of searches. Membership is required to overcome limitations. | | |

### 3.1.1.18 *Spiderfoot*

Spiderfoot[22] is a comprehensive reconnaissance tool. It gathers intelligence from more than 100 public data sources (open source intelligence – OSINT), collecting a multitude of elements that include IP addresses, domain names, e–mail addresses, names etc. A scan is created by picking the desired targets and the intelligence data to be gathered; a number of typical bundles of intelligence information is conveniently packed into respective use cases, while desired information can be tailored in detail by individually selecting specific items. Spiderfoot is available under GPL v2, some modules however need registration (and possibly payment) to work. Spiderfoot is mostly interactive, with limited possibilities for automation.

[21] https://dnsdumpster.com/
[22] https://www.spiderfoot.net

| | |
|---|---|
| Domain and subdomain names | X |
| IP addresses | X |
| Virtual hosts | X |
| Open ports, services, banners | X |
| Target spec | Domain, DNS name, IP address, IP subnetwork, email address |
| License | GPL v2 |

| | |
|---|---|
| E–mail addresses and peoples’ names | X |
| OS and version | X |
| Applications and their components | X |
| UI types | Web–based |
| Output options | CSV, GEXF |
| | |

#### *3.1.1.19 ReconDog*

ReconDog[23] is an open–source reconnaissance tool, made available under the Apache 2.0 license. It exploits external databases and locally driven searches to collect a multitude of information about its scan targets. It does not provide a graphical user interface, being command–line oriented. It is capable of collecting DNS and IP information, performing port scans or gathering the relevant information from the Censys.io databases, detecting web application technologies and CMSs, as well as identifying honeypots.

| | |
|---|---|
| Domain and subdomain names | X |
| IP addresses | X |
| Virtual hosts | – |
| Open ports, services, banners | X |
| Target spec | Domain, DNS name, IP address, IP subnetwork, URLs |
| License | Apache 2.0 |

| | |
|---|---|
| E–mail addresses and peoples’ names | – |
| OS and version | – |
| Applications and their components | X |
| UI types | Command line |
| Output options | CSV, GEXF |
| | |

### 3.1.2 Comparative analysis

In Table 3.3 and Table 3.4, we summarize the features of the network topology and host connectivity tools surveyed above.

Table 3.3. Network topology and host connectivity tools comparison (1/2)

| Tool | Active hosts | Reachability | Topology | OS & version | Active ports | Services and versions |
|---|---|---|---|---|---|---|

[23] https://github.com/s0md3v/ReconDog

| | | | | | | |
|---|---|---|---|---|---|---|
| **Nmap** | X | X | X | X | X | X |
| **Angry IP scanner** | X | X | X | X | X | X |
| **Unicornscan** | X | – | – | X | X | X |
| **Dipiscan** | X | X | X | X | – | X (limited) |
| **Masscan** | X | – | – | – | X | X |
| **Scanrand** | X | X | X (partial) | – | X | – |
| **Zmap** | X | – | – | – | X | X |
| **NetCrunch tools** | X | X | – | – | X | X |
| **MyNet toolset** | X | X | – | – | – | X |
| **LanTopoLog** | X | X | X | X | – | – |
| **Spiceworks NM** | X | X | X | X | X | X |
| **NetworkMiner** | X | – | – | X | X | X |
| **PcapViz** | – | X | X | – | – | – |
| **Skydrive** | – | X | X | – | – | – |

Table 3.4. Network topology and host connectivity tools comparison (2/2)

| Tool | UI & visualization | Offline result analysis | Output formats | License |
|---|---|---|---|---|
| **Nmap** | X (Zenmap & other tools) | X Active, online via (Zenmap) | Redirection of standard output, XML, Grepable, Script kiddie | GPL v2 |
| **Angry IP scanner** | X (Desktop UI) | Active scans only | CSV, TXT, XML, IP–Port list | GPL v2 |
| **Unicornscan** | – | Active scans only | Stdout redirection to log file, relational database, pcap file with received packets | GPL v2 |
| **Dipiscan** | X (Desktop UI) | Active scans only | Text files | freeware |
| **Masscan** | – | Active scans only | XML, binary, grepable, JSON, list | A–GPL–3 |
| **Scanrand** | – | Active scans only | Dump in database | BSD original |
| **Zmap** | – | Active scans only | Stdout redirection, CSV, Redis, JSON | Apache license v2 |
| **NetCrunch tools** | X | Active scans only | | adremsoft.com/netcrunch.tools/eula/ |

| | | | | |
|---|---|---|---|---|
| **MyNet toolset** | X | Active scans only | – (results cannot be saved) | Freeware |
| **LanTopoLog** | X | Active scans only | CSV | Shareware; the free version disables some features after a specific period of time. |
| **Spiceworks NM** | X (browser based) | Active scans only | A number of reports is available, which can be saved in CSV, XLS and PDF | Free after registration |
| **NetworkMiner** | X | Analysis of pcap files and passive scanning | Export to CSV / Excel / XML / CASE / JSON–LD (paid version only) | GPLv2; subscription option |
| **PcapViz** | X (GraphViz, dot) | Analysis of pcap files | Output redirection | N/A |
| **Skydrive** | X | Collection and analysis of log files | All facilities provided by Kibana and other Elastic search clients | Apache 2.0 |

According to the table above, NMAP and AngryIP scanner appear to be offering the most complete functionalities without any limitations, such as running on specific operating systems, licensing, fee requirement or hosting options. Both tools offer the capability to be extended, and, thus, cover more functionalities or be tailored to specific needs. Taking the above into account, these will be the tools that will be adopted for use in the context of Cyber–Trust. Both tools offer however limited capabilities for determining the network topology; these capabilities may be supplemented from other tools, such as NetworkMiner. Table 3.5 summarizes the features of the five reconnaissance tools reviewed in Section 3.1.1.

Table 3.5. Reconnaissance tools comparison

| | Maltego | Netglub | Dnsdumpster.com | Spiderfoot | ReconDog |
|---|---|---|---|---|---|
| **Domain and subdomain names** | X | X | X | X | X |
| **E–mail addresses and peoples' names** | X | X | – | X | – |
| **IP addresses** | X | X | X | X | X |
| **OS and version** | X | – | X | X | – |
| **Virtual hosts** | X | X | – | X | – |
| **Applications and their components** | X | – | X | X | X |
| **Open ports, services, banners** | X | – | X | X | X |

| | | | | | |
|---|---|---|---|---|---|
| **UI types** | Desktop | Desktop | Web–based | Web–based | Command line |
| **Target spec** | Domain, DNS Name, IPV4 Address, MX Record, NS Record, Autonomous System (AS), etc. | Domain, DNS name, IP address, IP subnetwork, URL, website, MX record, NS record, email address, person, phrase | Domain | Domain, DNS name, IP address, IP subnetwork, email | Domain, DNS name, IP address, IP subnetwork, URL |
| **Output options** | CSV, XLS, XLSX, PDF, image formats, GraphML, Entity Lists | CSV | XLSX | CSV, GEXF | Standard output, grepable |
| **License** | Community and paid editions | GPL v3 | Free | GPL v2 | Apache 2.0 |

Based on the information above, should a reconnaissance tool be needed in the context of Cyber–Trust for feeding the attack model, then the open source ReconDog seems to be a right option, whilst the Spiderfoot – up to the extent that its license limitations allow – will be also considered.

## 3.2 Vulnerability scanning

Vulnerability scanning is the process of assessing a network and its devices to discover vulnerable software or misconfigurations. The purpose of this process is to aware and to enable analysts or automated tools to take the necessary mitigation actions [100, 94]. In this section, first a review of vulnerability scanning and service discovery tool taxonomies is presented, along with existing vulnerability assessment standards, to aid in the choice of comparison criteria. Finally, existing vulnerability tools are examined for their suitability in the context of the Cyber–Trust.

### 3.2.1 Tools and scanning taxonomies

Vulnerability assessment methods can be classified as *manual*, *assistive*, and *fully automated* [62]. Manual assessments are performed by security analysts with domain knowledge and require a significant amount of time and resources to be committed. Towards the same direction, assistive methods are performed by security analysts using proper vulnerability scanning tools. On the other hand, fully automated methods are performed entirely by software. Mitigation for the first two categories is performed manually by security analysts, while the fully automated tools also automatically perform the necessary mitigation actions.

In this section, only tools allowing for a sufficient degree of automation will be covered. There are four types of vulnerability scanners [92]: (a) *port*, (b) *application*, (c) *host–based vulnerability*, and (d) *network–based vulnerability*. Specifically:

- *Port scanners* are used to discover open network ports of a network device and determine information about the services provided.
- *Application scanners* are used to assess the security state of a specific application or service.

- *Host–based vulnerability scanners* are used to assess the security state of the device they run on; having direct access to device resources enables them to better detect system misconfigurations, to consider attacks requiring local access and their findings can be more accurate than those of a network–based vulnerability scanner. They present scalability issues, since they need to be deployed and managed on each device separately.
- *Network–based vulnerability scanners* are used to assess the security state of the whole reachable (from the device they run on) network; having only network access to the systems to be assessed can present coverage problems as their service scanning module may miss network devices or services. Also, network disruptions may occur from the usage of such tools either by vulnerability tests, or even by normal service scanning (e.g., SCADA systems may misbehave while being scanned [19]).

In the context of vulnerability scanning, this section will cover tools under the last three categories, since the first category (port scanners) was covered in Section 3.1. Most application/vulnerability scanning tools include a service discovery module to provide information about the network devices (active hosts) and about the software/services they provide (service identification, OS fingerprinting) [94]. Service discovery techniques can be classified into *active probing* and *passive monitoring* [12].

- Active probing sends packages/messages to every service of each network device and analyses the response. This technique yields more complete results.
- Passive monitoring analyses captured network traffic to discover network services as they are used. Requires the installation of monitoring devices (specialized or general–purpose devices with the ability to capture network traffic) and the choice of monitoring points in the assessed network, a choice which can affect the analysis results. This technique is best used for trend analysis.

For both techniques, it is possible for network devices and services behind a firewall or network devices whose services are temporarily unavailable to be missed. Usage of application/vulnerability scanners presents some drawbacks, aside from those of their service discovery modules [92, 94]. The first drawback is that result inaccuracies may arise from malfunctioning user–created scripts/tests/plugins, incorrect identification of the network device services and their versions, and in some cases the need for the scanner to be authenticated to perform its assessment. Another drawback pertains to the reliance on a static knowledge base for performing vulnerability testing, which can make such tools miss zero–day vulnerabilities and if such a knowledge base remains outdated, they may also miss newer (known) vulnerabilities. A third drawback is that risk analysis is quite difficult to automate, since many tools consider the vulnerabilities in isolation, ignoring possible vulnerability combinations/correlations during a real–world attack (something that Cyber–Trust is taking into full consideration in order to devise intelligent mitigation strategies).

### 3.2.2 Comparison criteria choice

According to NIST [100], desired application/vulnerability scanner functionality includes: (a) enumeration of network devices; (b) discovery of software vulnerabilities and system/software misconfigurations; (c) the existence of knowledge base updating mechanism –in addition, information sources and their updating frequency should be considered; (d) automated analysis of the results to assess the security state of the network and its devices; (e) production of a structured/formatted report to be used by security analysts or other tools; and (f) use of open standards is strongly preferred, such as CVE (for vulnerability naming), OVAL (for testing the presence of a vulnerable software or service version), and CVSS (for vulnerability impact measurements). Alongside the desired functionality, the following should also be considered:

- Breadth (how many network devices or services are covered by the tool) and depth (how much information can be extracted for each network device or service) of the scanning operation.
- Third–party tool integration.
- Support for user–created scripts, tests or plugins.
- Tool license and usage restrictions.

The accuracy of the vulnerability scanning tools will not be considered since there is no standardized way of testing for false positives and false negatives. The comparison criteria for the tools listed in Section 3.2.3 are presented in Table 3.6.

Table 3.6. Comparison criteria of application/vulnerability scanning tools

| Field name | Field description | # values | Possible values |
|---|---|---|---|
| **Tool category** | The tool category from the taxonomy of vulnerability scanning tools [92] | ∞ | ▪ Application scanner<br>▪ Host–based vulnerability scanner<br>▪ Network–based vulnerability scanner |
| **Network device or service scanning method** | The category of the scanning module used by the tool from the taxonomy of scanning methods [12] | ∞ | ▪ Active probing<br>▪ Passive scanning<br>▪ Scanning is not supported (and textual description) |
| **Discovery of vulnerabilities and misconfigurations** | Whether the tool can only test software vulnerabilities and/or system misconfigurations | ∞ | ▪ Software vulnerabilities<br>▪ Software or system misconfigurations |
| **Breadth and depth of scanning** | Device or network coverage and types of devices and software assessed by the tool | ∞ | ▪ Complete network assessment (assessment of all discovered network devices)<br>▪ Complete network device assessment (assessment of all, or most services of a network device)<br>▪ Specific device assessment (and textual description)<br>▪ Specific application assessment (and textual description) |
| **Existence of knowledge base updating mechanism** | – | 1 | Yes/No and textual description |
| **Knowledge base information sources and update frequency** | – | 1 | Textual description |
| **Automated result analysis** | Ability to analyze the scanning results to derive more information about the security state of the network and its devices | 1 | Yes/No and textual description |
| **Output formats and their structure** | Each output format and its structure | ∞ | ▪ Structured – using open or publicly available standards<br>▪ Structured – using proprietary format |

| | | | |
|---|---|---|---|
| | | | ▪ Unstructured or textual |
| **Richness of the output report** | How much and what kinds of information are reported by the tool | 1 | Textual description |
| **Integration with third–party tools** | – | 1 | Textual description |
| **Interfacing options** | Existence of user interfaces, services and programming APIs | ∞ | ▪ Web Interface<br>▪ Graphical User Interface<br>▪ Console User Interface<br>▪ Application Programming Interface<br>▪ Other (and textual description) |
| **Support for user–added functionality** | Support for user–added functionality via user–created vulnerability tests and user–created plugins | ∞ | ▪ Support for user–created vulnerability tests and checks (and textual description)<br>▪ Support for user–added functionality (and textual description) |
| **License and usage restrictions** | – | 1 | Textual description |

*'∞' (resp. '1') means that multiple (resp. single) values are possible.*

### 3.2.3 List of tools considered

In the following subsections a non–exhaustive list of eighteen available (known) tools is presented; these are OpenVAS, Nessus, Nikto, Arachni, w3af, and Vega.

#### *3.2.3.1 OpenVAS*

The *Open vulnerability assessment system* (OpenVAS)[24] is a system of services and tools for network device vulnerability scanning. It consists of two main services: the OpenVAS Scanner, performing the *network vulnerability tests* (NVTs) and the OpenVAS Manager, controlling the OpenVAS Scanner as well as offering an *OpenVAS management protocol* (OMP) endpoint.

| | |
|---|---|
| **Tool category** | ▪ Network–based vulnerability scanner |
| **Network device or service scanning method** | ▪ Active probing |
| **Discovery of vulnerabilities and misconfigurations** | ▪ Software vulnerabilities<br>▪ Software or system misconfigurations |
| **Breadth and depth of scanning** | ▪ Complete network assessment<br>▪ Complete network device assessment |
| **Existence of knowledge base updating mechanism** | Yes |

[24] http://openvas.org/

| | |
|---|---|
| **Knowledge base information sources and update frequency** | Yes – the following feeds are provided that are updated daily:<br>▪ *Greenbone Community Feed* (GCF), is the default feed for OpenVAS. Contains over 50K *Network Vulnerability Tests* (NVTs).<br>○ Enterprise environments receive no updates since Sep. 2017<br>▪ *Greenbone Security Feed* (GSF), the commercial version of the GCF provided by Greenbone Security. |
| **Automated result analysis** | Yes – a prognostic scan can be performed to detect possible security issues without initiating a new scan.<br>If a scan has been performed more than once a vulnerability trend is also calculated and a delta report, containing only the difference between two reports, can be created and exported. |
| **Output formats and their structure** | Structured – using open or publicly available standards:<br>▪ XML<br>▪ XML – OVAL SC (System Characteristics) for each scanned system. Available via a custom reporting plugin provided by Greenbone.<br>▪ CSV – Containing only the discovered hosts, the CPE tables or the complete report.<br>▪ ARF – NIST Asset Reporting Format<br>Unstructured or textual:<br>▪ PDF – Detailing only the vulnerabilities or the complete report.<br>▪ LaTeX<br>▪ HTML<br>▪ TXT |
| **Richness of the output report** | For every identified vulnerability the following information is provided:<br>▪ CVE information, CVSS score and OVAL definition from the *National Vulnerability Database* (NVD).<br>▪ Related CERT advisories from the DFN–CERT and CERT–Bund. |
| **Integration with third–party tools** | Nmap, ike–scan, and debscan |
| **Interfacing options** | ▪ Web Interface, provided by the Greenbone Security Assistant component or any client supporting the OpenVAS Management Protocol – OMP.<br>▪ Console User Interface, provided by the OpenVAS CLI component.<br>▪ Other, directly with the OpenVAS Scanner and OpenVAS Manager services, as their communication protocols are documented. |
| **Support for user–added functionality** | Support for user–created vulnerability tests and checks:<br>▪ User–defined patterns for file content pattern matching.<br>▪ User–provided file checksums and checksum patterns.<br>▪ Custom CPE–based tests to detect the presence or absence of a specific class of applications or hardware.<br>Support for user–added functionality:<br>▪ Custom reporting plugins, to extract scan result information to custom or non–supported (by default) formats. |

| | |
|---|---|
| **License and usage restrictions** | Most components are licensed under the GNU GPL v2.0 and v3.0. For more information refer to the project repositories[25] |

### 3.2.3.2 *Nessus*

Nessus[26] is a network device vulnerability and configuration scanner. Vulnerability information is represented by scripts, referred to as plugins, written in the *nessus attack scripting language* (NASL).

| | |
|---|---|
| **Tool category** | Network–based vulnerability scanner |
| **Network device or service scanning method** | Active probing |
| **Discovery of vulnerabilities and misconfigurations** | ▪ Software vulnerabilities<br>▪ Software or system misconfigurations |
| **Breadth and depth of scanning** | Over 47K assets and network devices are covered (e.g. devices by HP, CISCO, etc.; operating systems, applications, device drivers, etc.).<br>▪ Complete network assessment<br>▪ Complete network device assessment |
| **Existence of knowledge base updating mechanism** | Yes |
| **Knowledge base information sources and update frequency** | More than 100K vulnerability tests, called plugins, covering over 45K CVE IDs and about 30K Bugtraq IDs are provided by Tenable. Over 100 new plugins per week are released. |
| **Automated result analysis** | Yes – the *Live Results* vulnerability scan can be performed to detect possible security issues without initiating a new scan. |
| **Output formats and their structure** | Structured – using open or publicly available standards:<br>▪ XML<br>▪ CSV<br>Structured – using proprietary format:<br>▪ NBE – Nessus report format, used by older Nessus versions; deprecated.<br>Unstructured or textual:<br>▪ HTML |
| **Richness of the output report** | For every identified vulnerability the following information is provided:<br>▪ Nessus plugin details:<br>○ Severity (Info/Low/Medium/High/Critical)<br>○ Nessus plugin ID and plugin version<br>○ Exploit type (e.g. Local), agent (e.g. Unix) and vulnerability test family (e.g. SuSE Local Security Checks)<br>▪ CVE ID, OSVDB ID, CVSS score, the affected software or assets in CPE format and others depending on the vulnerability. |

[25] https://github.com/greenbone

[26] https://www.tenable.com/products/nessus/nessus–professional

| | |
|---|---|
| | ▪ Synopsis, description and solution natural text fields, and related links.<br>▪ Nessus vulnerability test output and existing exploits/tools (e.g. Exploitable with: Metasploit) depending on the plugin/vulnerability. |
| **Integration with third–party tools** | Nmap, Nikto |
| **Interfacing options** | ▪ Web Interface<br>▪ Console User Interface, provided by the Nessus CLI utility; provides support for a subset of Nessus functionality (e.g. user management, updates, etc.) |
| **Support for user–added functionality** | Support for user–created vulnerability tests and checks:<br>▪ User–defined plugins (vulnerability tests) written in the *Nessus Attack Scripting Language* (NASL) |
| **License and usage restrictions** | Commercial license |

### *3.2.3.3 Nikto*

Nikto[27] is a web server vulnerability scanner with ability to check for misconfigurations and presence of insecure/outdated services, written in Perl. Nikto does not rely solely on the HTTP response codes as it uses the content of the response to check the presence of an indicator (file or specific content). The vendor claims that this significantly reduces false positives.

| **Tool category** | Application scanner |
|---|---|
| **Network device or service scanning method** | Scanning is not supported<br>▪ Multiple IP addresses, ports and URLs are specified in a text file.<br>▪ Nmap scan results can be piped as input to NIkto (e.g. nmap –p80 192.168.0.0/24 –oG – \| nikto.pl –h –). |
| **Discovery of vulnerabilities and misconfigurations** | ▪ Software vulnerabilities<br>▪ Software or system misconfigurations |
| **Breadth and depth of scanning** | Nikto is specialized to test web servers and web services. |
| **Existence of knowledge base updating mechanism** | Yes |
| **Knowledge base information sources and update frequency** | Tests are provided by CIRT Inc. for 6.7K dangerous files and programs, 1.25K outdated software version checks and 270 version–specific software checks. OSVDB (shut down since 2016) is the main source of information. |
| **Automated result analysis** | No |
| **Output formats and their structure** | Structured – using open or publicly available standards:<br>▪ XML<br>▪ CSV |

[27] https://cirt.net/nikto2

| | |
|---|---|
| | ▪ JSON – Saved request and response pairs.<br>Structured – using proprietary format:<br>▪ NBE – Nessus report format, used by older Nessus versions; deprecated.<br>Unstructured or textual:<br>▪ HTML<br>▪ TXT |
| **Richness of the output report** | Every vulnerability test contains the following fields:<br>▪ Test ID, used by Nikto.<br>▪ OSVDB ID<br>▪ Server type<br>▪ URI to retrieve<br>▪ HTTP method<br>▪ Strings to match.<br>▪ Summary, message to display when a test was successful.<br>▪ HTTP data, to send when using the POST method.<br>▪ Additional headers to send. |
| **Integration with third–party tools** | Can be launched by Nessus and results can be logged to Metasploit |
| **Interfacing options** | Console User Interface |
| **Support for user–added functionality** | Support for user–created vulnerability tests and checks:<br>▪ User–created tests for newer vulnerabilities.<br>Support for user–added functionality:<br>▪ User–created plugins for added functionality such as host detection, etc. |
| **License and usage restrictions** | Nikto is licensed under the GPL; tests are licensed for use with Nikto and require written permission from CIRT Inc. for other uses. |

#### *3.2.3.4 Arachni*

Arachni[28] is a web vulnerability scanning framework written in Ruby, specialized to test web servers, web services and web applications. A web browser environment is also implemented with support for standard web technologies (e.g. HTML5, JavaScript, AJAX), also supporting manipulation of the DOM and can simulate different browsing environment (e.g. by changing the user agent or the viewport). Arachni can tailor its vulnerability tests, referred to as checks, to the specific web application being tested and can train itself to follow and test new input vectors, allowing the assessment of complex web applications/pages.

| **Tool category** | Application scanner |
|---|---|
| **Network device or service scanning method** | Scanning is not supported<br>▪ The URL or IP address of the web application/server/page must be supplied by the user. |

[28] http://www.arachni–scanner.com

| | |
|---|---|
| **Discovery of vulnerabilities and misconfigurations** | ▪ Software vulnerabilities<br>▪ Software or system misconfigurations |
| **Breadth and depth of scanning** | Arachni is specialized to test web servers, web services and web applications. It can also perform OS vulnerability testing, tests on (commonly used in web applications) scripting languages (e.g. PHP, ASP, Python, Ruby, and the exception of Java) and tests on web frameworks (e.g. Rack, Rails, Django etc.) |
| **Existence of knowledge base updating mechanism** | No – vulnerability tests can be updated along with Arachni but not separately. |
| **Knowledge base information sources and update frequency** | Not applicable |
| **Automated result analysis** | No |
| **Output formats and their structure** | Structured – using open or publicly available standards:<br>▪ XML<br>▪ JSON<br>▪ YAML<br>Structured – using proprietary format:<br>▪ AFR – Arachni Framework Report format, the reference format for the reports created by Arachni. All other formats are based on the information contained in this report format.<br>Unstructured or textual:<br>▪ HTML<br>▪ TXT |
| **Richness of the output report** | A report contains:<br>▪ Screenshots of the web application and its DOM changes.<br>▪ HTML code of the DOM states.<br>▪ The flow of arguments through the JavaScript code.<br>▪ JavaScript execution snapshots, to capture injected JavaScript code.<br>▪ JavaScript execution context (stack, arguments, functions etc.).<br>▪ The HTTP requests and responses.<br>Each reported vulnerability contains the following information:<br>▪ Severity (Informational/Low/Medium/High)<br>▪ A textual description.<br>▪ Links to the corresponding data (as mentioned above). |
| **Integration with third–party tools** | No |
| **Interfacing options** | ▪ Web Interface<br>▪ Console User Interface<br>▪ Application Programming Interface: REST API<br>▪ Other: Ruby Library (as a Ruby gem) |

| | |
|---|---|
| **Support for user–added functionality** | Support for user–created vulnerability tests and checks:<br>▪ User–created vulnerability tests, referred to as checks.<br>Support for user–added functionality:<br>▪ User–created plugins to extend the functionality of Arachni.<br>▪ User–created report extractors, referred to as reporters, to export the scan report in any format. |
| **License and usage restrictions** | Arachni is licensed under the Arachni Public Source License[29]; restricted for commercial use, written permission is needed. |

### *3.2.3.5 w3af*

w3af[30] is a web application vulnerability scanning framework written in Python. It is comprised by three categories of modules: the *core modules* containing framework management modules and core libraries, the *user interface modules* and the *plugin modules* containing the rest of the w3af functionality, such as the fuzzing engine or the vulnerability checks. w3af also provides payloads and can perform exploitation of found vulnerabilities.

To perform a web application scan, w3af performs a three–phase process: first it indexes the whole web application using the available crawling plugins, then it tests the whole discovered application for possible vulnerabilities using the audit plugins, and then the results (and any error and debugging messages) are sent to the output plugins to be exported in the desired format. If exploitation is desired, then right after the audit plugins are finished, the attack plugins can be used to perform exploitation.

| **Tool category** | Application scanner |
|---|---|
| **Network device or service scanning method** | Scanning is not supported<br>▪ The URL or IP address of the web application must be supplied by the user. |
| **Discovery of vulnerabilities and misconfigurations** | Software vulnerabilities |
| **Breadth and depth of scanning** | w3af is specialized to test and (if desired) exploit web applications. |
| **Existence of knowledge base updating mechanism** | No – vulnerability tests can be updated along with w3af but not separately |
| **Knowledge base information sources and update frequency** | Not applicable |
| **Automated result analysis** | No |
| **Output formats and their structure** | Structured – using open or publicly available standards:<br>▪ XML<br>▪ CSV<br>Unstructured or textual:<br>▪ TXT<br>▪ HTML |

[29] http://www.arachni–scanner.com/license/
[30] http://w3af.org/

| | |
|---|---|
| **Richness of the output report** | The resulting report contains:<br>▪ A textual description of the vulnerability.<br>▪ The request and its corresponding response data. |
| **Integration with third–party tools** | No |
| **Interfacing options** | ▪ Graphical User Interface<br>▪ Console User Interface<br>▪ Application Programming Interface (REST API) |
| **Support for user–added functionality** | Support for user–created vulnerability tests and checks:<br>▪ User–created vulnerability tests are implemented as plugins and w3af supports user–created plugins.<br>Support for user–added functionality:<br>▪ Since w3af is a modular framework of reusable software components, addition of custom functionality is supported. |
| **License and usage restrictions** | w3af is licensed under the GPL 2.0. |

### *3.2.3.6 Vega*

Vega[31] is a GUI–based web application scanner written in Java. Along with its scanning capabilities an intercepting proxy (a program intercepts the traffic generated from the testing system and the system to be assessed allowing its user to study or modify it) is also included. The intercepting proxy can be used in conjunction with the automated testing capabilities of Vega to test the target application while the user is browsing it, thus achieving greater coverage.

| | |
|---|---|
| **Tool category** | Application scanner |
| **Network device or service scanning method** | Scanning is not supported<br>▪ The URL or IP address of the web application must be supplied by the user. |
| **Discovery of vulnerabilities and misconfigurations** | ▪ Software vulnerabilities<br>▪ Software or system misconfigurations |
| **Breadth and depth of scanning** | Vega is specialized to test web applications |
| **Existence of knowledge base updating mechanism** | No – vulnerability tests can be updated along with Vega but not separately. |
| **Knowledge base information sources and update frequency** | Not applicable |
| **Automated result analysis** | No |
| **Output formats and their structure** | Structured – using open or publicly available standards:<br>▪ XML alerts. |
| **Richness of the output report** | Both the XML alerts and the resulting report (as viewed from the GUI) contains: |

[31] https://subgraph.com/vega/

| | |
|---|---|
| | ▪ Classification and Severity of the vulnerability.<br>▪ The Impact of the vulnerability and recommended remediation steps (both are represented as lists).<br>▪ A natural text description of the vulnerability, referred to as the Discussion field.<br>▪ Reference links. |
| **Integration with third–party tools** | No |
| **Interfacing options** | ▪ Graphical User Interface |
| **Support for user–added functionality** | Support for user–created vulnerability tests and checks:<br>▪ User–created vulnerability tests are implemented as plugins and Vega supports user–created plugins.<br>Support for user–added functionality:<br>▪ Vega supports user–created plugins, also referred to as modules, written in JavaScript. |
| **License and usage restrictions** | Vega is licensed under the Eclipse Public License v1.0. |

### 3.2.4 Comparative analysis

Following is a summary of the information presented in Section 3.2.3, used to inform the choice of vulnerability scanning tools covering the needs of the Cyber–Trust project.

| | **OpenVAS** | **Nessus** | **Nikto** | **Arachni** | **w3af** | **Vega** |
|---|---|---|---|---|---|---|
| **Tool category** | Network–based Vulnerability Scanner | | Application Scanner | | | |
| **Network device or service scanning method** | Active Probing | | Not supported, IPs or URLs must be supplied by the user | | | |
| **Discovery of vulnerabilities and misconfigurations** | Both | | | | Vulnerabilities only | Both |
| **Breadth and depth of scanning** | Complete network and device assessment | | Web server and web service testing | Web server, web service and web application testing | Web application testing | |
| **Existence of knowledge base updating mechanism** | Yes | | | No | | |
| **Knowledge base information** | Two feeds updated daily, | Feed updated weekly, with | Feed based on OSVDB | Not applicable | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **sources and update frequency** | with over 50K vuln. tests | over 100K vuln. tests | (shut down on 2016) | | | |
| **Automated result analysis** | Yes | | No | | | |
| **Output formats** | XML, CSV, ARF, PDF, LaTeX, HTML, TXT | XML, CSV, HTML | XML, CSV, JSON, HTML, TXT | XML, JSON, YAML, AFR, HTML, TXT | XML, CSV, HTML, TXT | XML Alerts |
| **Richness of the output report** | CVE ID, CVSS score, OVAL definition, related CERT advisories | Severity, exploit type, exploit agent, CVE ID, OSVDB ID, CVSS score, CPE information, existing exploits, description and mitigation actions | OSVDB ID, server type, URI, HTTP method, summary | Severity, description, references and data used on the specific vuln. test | Description, requests with their corresponding data | Vulnerability classification, severity, impact, mitigation actions, description, references |
| **Integration with third–party tools** | NMap, ike–scan, debscan | NMap, Nikto | No | | | |
| **Interfacing options** | Web UI, CUI | | CUI | Web UI, CUI, API | GUI, CUI, API | GUI |
| **Support for user–defined tests and user–added plugins** | Both | User–defined tests | Both | | | |
| **License and usage restrictions** | GPL v2.0 & v3.0 | Commercial | GPL | APL, restricted for commercial use | GPL v2.0 | EPL v1.0 |

There were two main types of tools presented in Section 3.2.3: network–based vulnerability scanners designed to perform complete assessment of network devices, and application scanners specialized for web server/service/application testing. Two vulnerability scanning tools are recommended, one from each type, should the use of such tools be needed.

For the first type–network–based vulnerability scanners, the use of OpenVAS is recommended as it has already been used in numerous works (e.g., [4, 30]). It can output its results in highly structured and open formats, supports modifications (via user–created vulnerability tests, functionality plugins and even direct modifications), supports automation, and being open–source it has no usage or modification restrictions.

Finally, for the second type–application scanners, the use of Arachni is recommended as it covers the assessment of web servers, web services and web applications. It can output its results in highly structured and open formats, provides a variety of interfacing options (Web UI, Console UI and an API) and supports user–created vulnerability tests and functionality plugins; the only drawback is the requirement of written permission for Arachni to be used in a commercial product.

## 3.3 Exploit intelligence acquisition

Alongside network topology information (covered in Section 3.1) and discovered vulnerabilities for each network device (covered in Section 3.2), further information about the vulnerabilities is required in order to accurately model attacks and design mitigation schemes. This section presents a review of the existing methodologies for the extraction of the aforementioned information, a review of existing taxonomies with regards to security conditions (i.e. system's aspects dealing with its security state) and their relevance to the Cyber–Trust project, along with a comparison between the available vulnerability intelligence sources.

### 3.3.1 Pre/post–condition extraction

According to Aksu *et al.* [4], a common approach for generating graphical security model is the Pre/post–condition approach (also referred to as Prerequisite/Postcondition or Requires/Results–In). This requires quite detailed information about what should be satisfied in order to exploit a vulnerability (i.e. the pre–conditions), and the results of a successful vulnerability exploitation (i.e. the post–conditions).

Typically, pre–conditions include information going beyond the network connectivity of a network device or the reachability of the targeted service, such as the required privileges an attacker needs to have, the services provided by a network device, the specific versions of a vulnerable software, etc. On the other hand, post–conditions include information about the effects of a successful vulnerability exploitation, such as the resulting privileges of an attacker, the possibility of (arbitrary) code execution on the targeted system, the initiation of a *Denial of Service* (DoS) attack, etc.

The automated extraction of pre/post–condition information from exploit intelligence sources, such as vulnerability databases (e.g. the National Vulnerability Database) or other semi–structured or unstructured sources, remains an open problem [4] with many previous works on attack graph generation not covering the information extraction process. The remainder of this sub–section presents a review of related works with a focus on the information extraction process and various natural language processing methods used to construct the attack graphs.

#### *3.3.1.1 Aksu et al. (2018)*

The model proposed by Aksu et al. [4] uses information about the network topology, the existing vulnerabilities (from Nessus or OpenVAS) and information from the *National Vulnerability Database* (NVD) for the vulnerabilities themselves. Pre–conditions for an attack constitute the required location of an attacker on the network, referred to as the *access vector* (AV), and the privileges required to exploit a vulnerability. The results of a successful attack, i.e. the post–conditions, are the privileges acquired by the attacker. The particular information utilized for pre– and post–conditions are illustrated in Table 3.7.

Table 3.7. Pre/post–conditions used by [4]

| Pre–conditions | Post–conditions | Information sources |
|---|---|---|
| Privileges<br>▪ OS Admin<br>▪ OS User<br>▪ Virtualized OS Admin<br>▪ Virtualized OS User | Privileges<br>▪ OS Admin<br>▪ OS User<br>▪ Virtualized OS Admin<br>▪ Virtualized OS User | Network topology<br>▪ No specific tools mentioned.<br>Existing vulnerabilities<br>▪ Nessus or OpenVAS reports. |

| | | |
|---|---|---|
| ▪ Application Admin<br>▪ Application User<br>▪ None | ▪ Application Admin<br>▪ Application User<br>▪ None | Vulnerability intelligence<br>▪ National Vulnerability Database |

The AV is commonly obtained from the *common vulnerability scoring system* (CVSS) that is associated with a vulnerability, as documented in the *common vulnerabilities and exposures* (CVE) items of vulnerability databases. The values taken by the AV are: (a) physical, (b) local, (c) adjacent network, and (d) network. Two methods of privilege generation from the NVD description text were tested: a rule–based, using a reasoning engine and manually created rules, and one using *machine learning* (ML).

#### *3.3.1.2 Gosh et al. (2015)*

Cyber–Trust, a tool presented by Gosh et al. [30] in 2015, uses information about the network topology (using manually entered information, firewall rules and the OpenVAS report), the existing vulnerabilities (from the OpenVAS report), and information for the available exploits for each identified vulnerability from the Metasploit framework[32] exploit modules (if the required information does not exist, the Open Source Vulnerability Database and the Bugtraq[33] exploit description is used).

Pre–conditions for an attack are: the existence of a vulnerability on a network device, the attacker's connectivity to the targeted network device and the required privileges. Post–conditions are not specified as they are generated by the tool at runtime considering the reported vulnerabilities and the available exploits.

Table 3.8. Pre/post–conditions used by [30]

| Pre–conditions | Post–conditions | Information sources |
|---|---|---|
| ▪ Existence of a specific vulnerability<br>▪ Existence of a vulnerable software version<br>▪ Existence of a specific architecture<br>▪ Connectivity with target<br>▪ Privileges | ▪ Metasploit modules to extract information via keywords and key–phrases<br>▪ OSVDB and Bugtraq descriptions | Network topology<br>▪ Manually entered information<br>▪ Firewall rules<br>▪ OpenVAS report<br>Existing vulnerabilities<br>▪ OpenVAS report<br>Vulnerability intelligence<br>▪ Metasploit exploit modules<br>▪ OSVDB and Bugtraq descriptions |

#### *3.3.1.3 Weerawardhana et al. (2015)*

Weerawardhana *et al.* [152] tested two methods to extract the required information from the NVD for the generation of *personalized attack graphs* (PAGs); one using a machine learning approach and another using a part–of–speech tagging engine. PAGs, which are described in [148], need information about the target system (existing vulnerabilities, system configuration, access privileges), the actions of the user (user system configuration, user habits or activities, sensitive information to be protected) and the actions that an attacker has to perform for conducting a successful attack. The extracted information includes software names and versions, file names, type of a vulnerability, user and attacker actions (as defined by the PAG), and impacts.

[32] https://www.metasploit.com/
[33] http://bugtraq–team.com/

#### 3.3.1.4 *Joshi et al. (2013)*

Joshi *et al.* [58] proposed a method for the conversion of semi–structured or unstructured vulnerability information from the NVD to an RDF format. The tool uses an entity and concept spotter to classify textual terms in the following categories: software and OS (existence of a specific software application and in some cases its version), network terms (e.g. IP address, SSL, etc.), attack means (a method of attack, e.g. buffer overflow) and attack consequences (e.g. denial of service), file name, hardware, *named entity recognition* (NER) modifier (follows the software and OS categories, specifies a range of versions, e.g. Adobe Acrobat X *and earlier versions*), and other technical terms.

#### 3.3.1.5 *Roschke et al. (2009)*

Roschke *et al.* [123] presented one of the earliest works specifically aimed at information extraction from *vulnerability databases* (VDBs) for attack graph generation. A data model was proposed to unify vulnerability information from different VDBs using both the available semi–structured information and information extracted from the vulnerability description. An add–on module for the MulVAL system (*see* [109] and Section 5 for more details) was also implemented to test the effectiveness of their data model. A comparative analysis of ten VDBs led to the selection of seventeen fields conveying highly relevant and useful information (if available from the VDB fields); these are provided in Table 3.9.

Table 3.9. Relevant fields of vulnerability information in [123]

| Relevant fields | Relevant fields |
|---|---|
| 1. Vulnerability title<br>2. Vulnerability description<br>3. CVE ID<br>4. Vendor–specific ID<br>5. Publication date<br>6. Date of last update<br>7. Popularity<br>8. Person/entity who discovered the vulnerability<br>9. Range, position of the attacker on the network for the vulnerability to be exploitable | 10. Affected OS and other software, and their affected versions<br>11. CVSS score<br>12. Complexity of exploitation<br>13. Required authentication/privileges for the vulnerability to be exploitable<br>14. Impact of vulnerability<br>15. References<br>16. Mitigation measures/actions<br>17. Vulnerability status (e.g. fixed or not) |

The authors consider the items 9–13 useful to determine the pre–conditions of a vulnerability and the item 14 (the impact of a vulnerability) useful to determine its post–conditions. The items 5, 6 (publication date and date of last update) are used to determine if an updated version of the VDB entry is available.

Table 3.10. Pre/post–conditions used by [123]

| Pre–conditions | Post–conditions | Information sources |
|---|---|---|
| Extracted from:<br>▪ Item 9: range<br>▪ Item 10: affected OS and software (with their versions)<br>▪ Item 11: CVSS score | Extracted from:<br>▪ Item 14: Impact of vulnerability | Vulnerability intelligence:<br>▪ From various VDBs (10 were tested by the authors) |

| | | |
|---|---|---|
| ▪ Item 12: complexity of exploitation<br>▪ Item 13: required privileges or authentication | | |

The proposed data model representing information extracted from the vulnerability description consists of three related properties:

- *System properties* representing system characteristics, such as the existence of specific accounts or a specific software/OS version.
- *Influence properties*, representing changes on the system properties after successful exploitation of the vulnerability.
- *Range properties*, representing the location of the attacker on the network for a vulnerability to be exploited.

For the influence properties two types of resources are considered: *passive* (e.g. files or database data) and *active* (e.g. services or running software), and specific actions are mapped to loss of confidentiality, integrity or availability. More precisely, read access, write access, and deletion/destruction of passive resources are mapped to loss of confidentiality, integrity, and availability respectively; on the other hand, influencing the output and losing a service's existence in active resources were mapped to loss of integrity and availability respectively.

### 3.3.2 Relations with CWE

The *Common Weakness Enumeration*[34] (CWE) is a formal list of security vulnerabilities and other security weaknesses maintained by the MITRE corporation; developed alongside the CVE list, CWE can be used to map potential weaknesses and vulnerabilities with their observed instances.

Mapping a discovered weakness to its CWE concept, in the context of Cyber–Trust, can aid in the choice of mitigation actions, and add high–level information about a vulnerability and its causes. Each of the 716 weakness entries of the CWE list can be classified as a:

- *Class weakness*–described in the most abstract terms (e.g. *CWE–697: Incorrect Comparison*).
- *Base weakness*–described with enough details to be detectable and mitigated while still being abstract (e.g. *CWE–1025: Comparison Using Wrong Factors*).
- *Variant weakness*–the most detailed description containing low–level technology–specific details (e.g. *CWE–595: Comparison of Object References Instead of Object Contents*).
- *Composite weakness*–a group of two or more weaknesses that need to be present at the same time for a vulnerability to be present (e.g. *CWE–689: Permission Race Condition During Resource Copy* requires both *CWE–362: Concurrent Execution using Shared Resource with Improper Synchronization (Race Condition)* and *CWE–732: Incorrect Permission Assignment for Critical Resource* to be present).

A weakness entry may also be related with other weakness entries via child–of/parent–of relations (e.g. in the research concepts view CWE–595 is a child of CWE–1025) and weakness entries sharing common characteristics can be grouped under *categories* (with over 200 categories existing in the CWE list). Each entry contains the information depicted in Table 3.11.

[34] https://cwe.mitre.org/

Table 3.11. CWE entry fields

| CWE entry field | CWE entry field |
|---|---|
| CWE identifier | Possible mitigation actions |
| Name and description | Node relationship (child–of/parent–of relations) |
| Alternate terms | Source taxonomies |
| Description of the behavior | Code samples for weaknesses pertaining to a specific language or architecture |
| Description of the exploit | CVE identifier |
| Likelihood of exploit existence/creation | References |
| Description of the consequences of successful exploitation | |

Weakness entries (either by themselves or in Categories) can be viewed through 32 hierarchical representations, referred to as *Views*, with the three most significant being: the *Research Concepts View*, the *Development Concepts View* and the *Architectural Concepts View*. The remainder of this section presents a high–level review of these three views; more detailed information can be viewed directly from the CWE definitions.

The Research Concepts View (CWE–1000[35]) is aimed at academic researchers, vulnerability analysts and assessment vendors (to test their vulnerability detection tools) and presents all 716 weakness entries organized according to abstractions in software behaviors. Table 3.12 presents the top–level entries, also referred to as *Pillars*.

Table 3.12. Top–level entries included in the Research Concepts View (CWE–1000)

| CWE ID | Title | CWE ID | Title |
|---|---|---|---|
| CWE–682 | Incorrect Calculation | CWE–693 | Protection Mechanism Failure |
| CWE–118 | Incorrect Access of Indexable Resource (Range Error) | CWE–697 | Incorrect Comparison |
| CWE–330 | Use of Insufficiently Random Values | CWE–703 | Improper Check or Handling of Exceptional Conditions |
| CWE–435 | Improper Interaction Between Multiple Correctly–Behaving Entities | CWE–707 | Improper Enforcement of Message or Data Structure |
| CWE–664 | Improper Control of a Resource Through its Lifetime | CWE–710 | Improper Adherence to Coding Standards |
| CWE–691 | Insufficient Control Flow Management | | |

[35] https://cwe.mitre.org/data/definitions/1000.html

The Development Concepts View (CWE–699[36]) is aimed at software developers and educators, presenting 708 of the 716 weakness entries and 42 of the 247 total categories in the CWE, covering concepts used in software development. Table 3.13 presents the top–level entries.

Table 3.13. Top–level entries included in the Development Concepts View (CWE–699)

| CWE ID | Title | CWE ID | Title |
|---|---|---|---|
| CWE–16 | Configuration | CWE–840 | Business Logic Errors |
| CWE–19 | Data Processing Errors | CWE–442 | Web Problems |
| CWE–21 | Pathname Traversal and Equivalence Errors | CWE–355 | User Interface Security Issues |
| CWE–189 | Numeric Errors | CWE–452 | Initialization and Cleanup Errors |
| CWE–254 | 7PK – Security Features | CWE–465 | Pointer Issues |
| CWE–361 | 7PK – Time and State | CWE–490 | Mobile Code Issues |
| CWE–389 | Error Conditions, Return Values, Status Codes | CWE–559 | Often Misused: Arguments and Parameters |
| CWE–399 | Resource Management Errors | CWE–569 | Expression Issues |
| CWE–417 | Channel and Path Errors | CWE–657 | Violation of Secure Design Principles |
| CWE–429 | Handler Errors | CWE–1006 | Bad Coding Practices |
| CWE–438 | Behavioral Problems | | |

*7PK refers to the 'Seven Pernicious Kingdoms' (CWE–700) category, based on [146].*

The Architectural Concepts View (CWE–1008[37]) is aimed at software designers and educators, presenting 223 of the 716 weakness entries and 42 of the 247 categories, organizing them according to common architectural security tactics. Table 3.14 presents the top–level entries.

Table 3.14. Top–level entries included in the architectural concepts view (CWE–1008)

| CWE ID | Title | CWE ID | Title |
|---|---|---|---|
| CWE–1009 | Audit | CWE–1015 | Limit Access |
| CWE–1010 | Authenticate Actors | CWE–1016 | Limit Exposure |
| CWE–1011 | Authorize Actors | CWE–1017 | Lock Computer |
| CWE–1012 | Cross Cutting | CWE–1018 | Manage User Sessions |
| CWE–1013 | Encrypt Data | CWE–1019 | Validate Inputs |
| CWE–1014 | Identify Actors | CWE–1020 | Verify Message Integrity |

[36] https://cwe.mitre.org/data/definitions/699.html
[37] https://cwe.mitre.org/data/definitions/1008.html

### 3.3.3 Vulnerability intelligence sources

This section presents a review of vulnerability intelligence sources, that will be taken with a focus on semi–structured *vulnerability databases* (VDBs)[38]; the comparison criteria used are those illustrated in Table 3.9 except the following fields: *Popularity*, *Exploitation complexity*, *Required authentication or privileges* and *Vulnerability status*, as none of the reviewed VDBs contain such information. Additional information about the usage of standards such as the *Common Platform Enumeration* (CPE) or *Common Weakness Enumeration* (CWE) and available formats are also considered.

The fields in the following comparative analysis refer to information existing in specific fields of the VDBs and not on information that can be extracted from them. If no information about the license or usage restrictions is reported, it is assumed that the maintainer holds the copyright to the information in the VDB. A comparative analysis of the available VDBs is conducted in the following tables, i.e. Table 3.15, Table 3.16, and Table 3.17.

Table 3.15. Comparative analysis of VDBs (1/3)

| | Maintainer | Size | License | Vuln. title | Vuln. details | Available formats |
|---|---|---|---|---|---|---|
| **Nat'l Vulnerability Database (NVD)[41]** | National Institute of Standards and Tech. (NIST) | ≥ 115K | Public domain | – | X | XML, JSON, HTML, RSS feed |
| **Rapid7 Vulnerability & Exploit DB[42]** | Rapid7 | ≥ 70K | – | X | X | HTML |
| **Security Focus DB[43]** | SecurityFocus | – | Copyright held by the maintainer | X | X | HTML |
| **Exploit DB[44]** | Offensive Security | ≥ 40K | GPL v2.0 | X | X | HTML, RSS feed, Raw data on GitHub[39] |
| **AusCERT Security Bulletins[45]** | AusCERT, at Univ. of Queensland | – | Copyright held by the maintainer | X | X | HTML, RSS feed |
| **CERT/CC Vulnerability Notes DB[46]** | CERT/CC, at Carnegie Mellon Univ. | – | Permission required for any use | X | X | HTML, RSS feed, Incomplete data on GitHub[40] |
| **Common Vulnerabilities & Exposures[47]** | MITRE Corporation | ≥ 110K | Permission granted s.t. conditions | – | X | HTML, CVRF |
| **ICS–CERT Advisories[48]** | NCCIC, U.S. Dept. Homeland Security | – | – | X | X | HTML, RSS feed |

[38] https://first.org/global/sigs/vrdx/vdb–catalog/
[39] https://github.com/offensive–security/exploitdb
[40] https://github.com/CERTCC/Vulnerability–Data–Archive

| | | | | | | |
|---|---|---|---|---|---|---|
| **Japan Vulnerability Notes (JVN)**[49] | JPCERT/CC and IPA | – | Copyright held by the maintainer | X | X | HTML, RSS feed |
| **JVN iPedia**[50] | Information technology Promotion Agency (IPA) | – | Copyright held by the maintainer | X | X | HTML, RSS feed, VULDEF (XML–based), API |
| **JC3 Bulletin Archive**[51] | U.S. Dept. of Energy | – | – | X | X | HTML, RSS feed |
| **NCSC–FI Vulnerability Database**[52] | Finnish Communications Regulatory Authority | – | – | X | X | HTML |
| **VulDB**[53] | VulDB | ≥ 125K | Creative Commons CC BY–NC–SA 4.0 | X | X | HTML, RSS feed, API |
| **SecurityTracker**[54] | SecurityGlobal.netLLC | – | Copyright held by the maintainer | X | X | HTML |
| **TippingPoint Zero Day Initiative**[55] | Trend Micro | – | – | X | X | HTML, RSS feed |

Table 3.16. Comparative analysis of VDBs (2/3)

| | CVE ID | Vendor–specific ID | CVSS score | CWE use | CPE use | Affected H/W, S/W |
|---|---|---|---|---|---|---|
| **Nat'l Vulnerability Database (NVD)**[41] | X | – | X | X | X | X |
| **Rapid7 Vulnerability & Exploit DB**[42] | X | – | X | – | – | X |
| **Security Focus DB**[43] | X | X | – | – | – | X |
| **Exploit DB**[44] | X | X | – | – | – | X |
| **AusCERT Security Bulletins**[45] | X | X | – | – | – | X |
| **CERT/CC Vulnerability Notes DB**[46] | X | X | X | X | – | X |
| **Common Vulnerabilities & Exposures**[47] | X | – | – | – | – | – |
| **ICS–CERT Advisories**[48] | X | X | X | X | – | X |
| **Japan Vulnerability Notes (JVN)**[49] | X | X | X | X | – | X |

| | | | | | | |
|---|---|---|---|---|---|---|
| JVN iPedia[50] | X | X | X | X | – | X |
| JC3 Bulletin Archive[51] | X | X | – | – | – | – |
| NCSC–FI Vulnerability DB[52] | X | X | – | – | – | X |
| VulDB[53] | X | X | X | X | X | X |
| SecurityTracker[54] | X | X | – | – | – | X |
| TippingPoint Zero Day Initiative[55] | X | X | X | – | – | X |

Table 3.17. Comparative analysis of VDBs (3/3)

| | Impact | Credit | Range | Publication date | Last upd. date | References |
|---|---|---|---|---|---|---|
| Nat'l Vulnerability Database (NVD)[41] | – | X | – | X | X | X |
| Rapid7 Vulnerability & Exploit DB[42] | – | – | – | X | X | X |
| Security Focus DB[43] | X | X | X | X | X | X |
| Exploit DB[44] | – | X | – | X | – | – |
| AusCERT Security Bulletins[45] | X | – | – | X | – | X |
| CERT/CC Vulnerability Notes DB[46] | X | X | – | X | X | X |
| Common Vulnerabilities & Exposures[47] | – | X | – | X | X (in title) | X |
| ICS–CERT Advisories[48] | X | X | – | X | – | X |
| Japan Vulnerability Notes (JVN)[49] | X | X | – | X | X | X |
| JVN iPedia[50] | X | – | – | X | X | X |

[41] https://nvd.nist.gov/
[42] https://www.rapid7.com/db/
[43] https://www.securityfocus.com/bid/
[44] https://www.exploit–db.com/
[45] https://www.auscert.org.au/bulletins/
[46] https://www.kb.cert.org/vuls/
[47] http://cve.mitre.org/
[48] https://ics–cert.us–cert.gov/advisories/
[49] http://jvn.jp/en/
[50] https://jvndb.jvn.jp/en/

| | | | | | | |
|---|---|---|---|---|---|---|
| **JC3 Bulletin Archive**[51] | X | – | – | X | – | – |
| **NCSC–FI Vulnerability DB**[52] | X | X | X | X | X | X |
| **VulDB**[53] | X | – | X | X | X | X |
| **SecurityTracker**[54] | X | – | – | X | – | X |
| **TippingPoint Zero Day Initiative**[55] | X | X | – | X | X | – |

From the comparative analysis presented in the above tables and for the primary vulnerability information source, NVD that is maintained by NIST is the most complete one, its information is in the public domain– and thus can be used without restriction. In addition, it uses open standards for many of its fields (CVE IDs – allowing links with other VDBs, CVSS scores, CWE and CPE information) and its information is available in many structured and open formats (XML, JSON along with HTML and an RSS feed). In addition, the Exploit Database also contains useful information, as it maintains exploit code that may be useful in testing the vulnerability in question or for conducting further analysis.

Several tools for information retrieval have been presented, a non–comprehensive selection of four tools will be presented in the remainder of this section.

- CVE–Search[56] is a tool for local storage and offline access to CVE and CPE information, written in Python 3 and using MongoDB for information storage. It utilizes the NVD, CVE and the Microsoft Security Bulletins for vulnerability information, and for exploit code it utilizes the Exploit Database and the D2 Elliot Web Exploitation Framework[57] data.
- CVE–Scan[58] combines the results of an Nmap scan (run manually by the user) with CVE–Search to perform a simple vulnerability scan of the network. CVE–Search is licensed under the GNU Affero GPL v3.0 and CVE–Scan under the Original BSD license.
- SearchSploit[59] is a tool maintained by Offensive Security for their Kali Linux penetration testing distribution allowing offline searches to the Exploit Database. SearchSploit is licensed under the GPL v2.0.
- Stucco[60] is a suite of tools for the creation of knowledge graphs from various unstructured and semi–structured information sources, like VDBs and various program logs. Three modules[61] were implemented for the retrieval of information from semi–structured VDBs: for the NVD, Bugtraq DB and Sophos RSS feed, with the last two being deprecated. Stucco is licensed under the MIT license.

[51] https://www.energy.gov/articles/673/708757+708775/JC3 Bulletin Archive
[52] https://www.viestintavirasto.fi/en/cybersecurity/vulnerabilities.html
[53] https://vuldb.com/
[54] https://securitytracker.com/
[55] https://www.zerodayinitiative.com/advisories/published/
[56] https://cve–search.github.io/cve–search/
[57] https://www.d2sec.com/
[58] https://github.com/NorthernSec/cve–scan
[59] https://github.com/offensive–security/exploitdb
[60] https://stucco.github.io/
[61] https://github.com/stucco/collectors

The sources having been reviewed in this section will prove to be valuable towards sharing complete and accurate cyber–threat intelligence via the *enriched vulnerability database* (eVDB) that will also include rich information identified by Cyber–Trust's crawling service from the surface/deep web.

## 3.4 Information acquisition for attack mitigation

Attack mitigation refers to the methods and techniques that can be employed to contain and reduce the negative impacts of attacks on an infrastructure or service[62]. Another working definition of mitigation is *"the elimination or reduction of the frequency, magnitude, or severity of exposure to risks, or minimization of the potential impact of a threat or warning"*[63]. According to the NIST model [150] mitigation actions may be classified as *proactive* (i.e. taking place before an attack occurs, to tackle related vulnerabilities, reduce the attack surface or lessen the foreseen impact, should an attack occur) and *reactive* (i.e. taking place when an attack is detected, typically to stop the attack process). NIST [150] also defines a classification scheme for attack mitigation actions according to the nature of the actions taken follows:

- configure (adjust target configuration/settings)
- disable (turn off or uninstall a target component)
- enable (turn on or install a target component)
- patch (apply a patch, hotfix, update, etc.)
- policy (remediation requires out–of–band adjustments to policies or procedures)
- restrict (adjust permissions, access rights, filters, or other access restrictions)
- update (install upgrade or update the system)
- combination (combination of two or more approaches)

Out of these mitigation action categories, *policy* refers to activities that concern procedures, practices and actions that are enforced outside of the narrow scope of the system to be protected, and henceforth will not be considered further. Considering the remaining action categories, *patch* and *update* are proactive actions, while *configure*, *disable*, *enable*, and *restrict* can be either proactive or reactive.

The objective of this subsection is to identify information sources that list mitigation actions that can be applied to tackle threats, combined with methods which enable the automated extraction of these actions. Besides the identification of actions, additional information that is useful in the context of attack mitigation will be considered: this information primarily concerns the impact that each mitigation action has on the value of each asset, an aspect that needs to be considered when selecting among possible mitigation actions to be applied. For example, in order to mitigate an information exfiltration attack to a service originating from a specific IP, it is clearly possible to shut down the service (a *disable* action); if the service configuration allows the specification of blacklisted IPs, it is possible to blacklist the IP from which the attack originates; and in the presence of a firewall appliance or some other IP–based access control (e.g. TCP wrappers) it is also possible to block the access to the service from the particular IP address. Although all choices clearly inhibit information exfiltration, it is also clear that the first mitigation method (service disablement) has a severe impact on the availability dimension of the asset and therefore one of the two remaining methods should be chosen whenever possible. Taking this aspect into account, we will also consider the identification and extraction of information regarding the impact on the organizational assets' value, which can be used to drive the mitigation action selection process.

### 3.4.1 Product and vendor–oriented security advisories

Product and vendor–oriented security advisories are catalogues hosting information about vulnerabilities that have been identified for specific products, coupled with specific instructions on how to mitigate these –

[62] https://www.ovh.com/asia/anti–ddos/mitigation.xml
[63] http://www.businessdictionary.com/definition/mitigation.html

whenever such instructions are available. An indicative list of security advisory databases is shown in the following table:

Table 3.18. Indicative list of security advisory databases

| Description | URL |
|---|---|
| Debian security advisory database | https://www.debian.org/security/2018/dsa–4332 |
| Microsoft security update summary | https://portal.msrc.microsoft.com/en–us/security–guidance/summary |
| Red Hat security advisories | https://access.redhat.com/security/security–updates/#/ |
| IBM security buletins | https://www.ibm.com/security/secure–engineering/bulletins.html |
| PHP security advisories | https://github.com/FriendsOfPHP/security–advisories |
| Ruby | https://github.com/rubysec/ruby–advisory–db |
| nodeJS | https://github.com/nodejs/security–wg/blob/master/processes/vuln_db.md |
| MariaDB | https://mariadb.com/kb/en/library/security/ |
| Huawei security advisories | https://www.huawei.com/en/psirt/all–bulletins |
| Android security bulletins | https://source.android.com/security/bulletin/2018–12–01.html |

Information within these databases is fairly structured, listing the precise package(s) that are covered by each security advisory, the vulnerabilities exhibited by these software packages (typically as references to CVE entries) and the mitigation actions that can be applied, usually in the form of patches/updates to be installed or configurations to be performed. The affected packages are listed in human–readable textual formats, and additionally using the software name and software versioning encoding scheme endorsed by the vendor (e.g. official product names and versions in the Microsoft security update, package names bundled with version information in Debian security advisory database and so forth), hence this information can be harvested to be later matched against the corresponding installed product information, when mitigation actions for a specific machine should be applied. The mitigation actions themselves, as stated above, mainly fall under the *patch*, *update* and *configure* categories.

Product–oriented security advisory databases have always a structured format, reflecting the information fields that are used to model an advisory. In some cases, it is possible to download the database in a format that is friendly to mechanized processing (e.g. JSON or XML documents), whereas in other cases only human–oriented formats (predominantly HTML pages) are available. In the latter case, since these HTML pages are highly structured, simple structure analysis of the pages and textual/pattern matching are sufficient to identify the mitigation actions. In the former case (i.e. database availability in mechanized processing–friendly formats), it suffices to extract and process the relevant fields, however in all cases a specific adapter to map the database–specific information schema to a unified Cyber–Trust information schema is needed.

Regarding patch and update file identification, this data can be extracted easily through structure analysis of the information and/or regular expression level matching. Furthermore, in most cases the installation of a patch is performed by executing the patch binary or overwriting the vulnerable package with an updated version, hence patch installation can be automated to a considerable extent.

Information about configuration changes that should be applied to mitigate an attack has a greater degree of variability, since the methods that can be used to apply the configuration changes are highly dependent on the product[64,65]. Therefore, converting configuration change information to actionable specifications is highly likely to require human expert intervention.

Disabling and/or uninstalling the software is highly automatable, since the official product/package name is included in the database entry.

Regarding additional information needed to perform attack mitigation, references to CVE entries are sufficient for obtaining information about aspects such as the impact, exploitability, attack vector and complexity of the threat; some advisory databases include local copies of these data, removing the necessity for an additional lookup. Installation of a patch and application of a configuration usually have a low impact at the availability of services (through the necessitation of service or machine restarts). On the other hand, disabling a service or removing the respective software effectively zeroes the availability score.

### 3.4.2 Generic security advisories and vulnerability databases

Besides product and vendor–oriented security advisories, security–focused organizations provide comprehensive lists of vulnerabilities that may affect any software or hardware asset, regardless of its vendor. A comprehensive list of these databases is included in subsection 3.3. The entries within these databases list the products (software and/or hardware, together with their versions) affected by the relevant vulnerability and the mitigation actions to be performed, whenever such information is available. However, comparing to the case of product and vendor–oriented security advisories, two major additional challenges exist towards the direction of turning the information in the database entries into actionable rules:

1. *Unambiguous and automated identification of the assets affected by the vulnerability.* While generic security advisories and vulnerability databases do refer to the assets that are affected by each vulnerability, the naming used to make these references does not correspond to the one endorsed by product vendor; this is also true for the versioning scheme. The different vocabularies and encoding schemes hinder the process of matching vulnerability database entries to organizational assets that need to be protected.

   In order to tackle this issue, a number of options are available, depending on the additional information present in the CVE:

   a. *Use of CPE information*: Common Platform Enumeration (CPE)[66] identifiers are used to precisely specify a platform (firmware, operating system, application software, container). Whenever such information is available in the vulnerability database *and* within the assets, the matching procedure to identify affected assets can be performed using CPE identifiers. Some vulnerability databases (e.g. NVD) include CPE information in their entries.

   b. *Use of SWID information*: Software identification (SWID) identifiers[67] are pointers to software identification documents. A SWID tag document is composed of a structured set of data elements that identify the software product, characterize the product's version, the organizations and individuals that had a role in the production and distribution of the product, information about the artifacts that comprise a software product, relationships between software products, and other descriptive metadata. The information in a SWID tag provides software asset management and security tools with valuable information needed to automate the management of a software install across the software's deployment lifecycle. SWID tags support automation of software inventory as part of a software asset management (SAM) process, assessment of software vulnerabilities present on a computing device, detection of missing patches, targeting of configuration checklist assessments,

[64] https://www.debian.org/security/2018/dsa–4112
[65] https://docs.microsoft.com/en–us/security–updates/securityadvisories/2016/3174644
[66] https://nvd.nist.gov/products/cpe
[67] https://nvd.nist.gov/products/swid

software integrity checking, installation and execution whitelists/blacklists, and other security and operational use cases.

SWID tags are currently supported on major OS platforms, including Windows, MacOS and Linux[68] and recommendations have been made to modify the vulnerability databases schema, replacing CPE tags with SWID identifiers [149], insofar however no vulnerability database has been found to list SWID identifiers.

2. *Identification of the mitigation instruction information*. In many cases, generic vulnerability databases provide mitigation instructions through references to vendors' web pages. A first issue encountered in this context is that references including mitigation actions are not clearly distinguishable from other references that simply confirm the existence of the vulnerability or provide other, not mitigation–related information. Furthermore, even in cases that the links can be distinguished (e.g. through associated tags or by having been structurally placed in a corresponding, clearly identifiable section of the document), the content of the links' target document exhibits a high degree of structural and content variability (due to the fact that it is provided by diverse authors), hence while it can be used for information harvesting, the degree of automation that can be supported at processing and application/enforcement level is limited.

In the following, we discuss on the above properties that relate to the content of the vulnerability databases listed in subsection 3.3.

***NVD:*** Within NVD, each CVE entry contains resource specifications in the form of URLs, and each such resource is characterized with a set of tags; out of all tag values, *Patch*, *Third Party Advisory*, *VDB Entry* and *Vendor Advisory* indicate that the associated URL resource points to a web page encompassing some mitigation option. The resource URLs typically point to human–readable web pages (as contrasted to highly structured documents like JSON or XML documents), and their content has a diverse format, since they are provided by different organizations. However these documents are structured with mitigation options appearing under suitable headings (e.g. *Solution*, *Workaround*, *Remediation/Fixes, Workarounds and mitigations*, therefore it is feasible to extract such information, albeit in many cases the extracted content cannot be used for fully automated determination of actions to be taken. NVD includes CPE information allowing each vulnerability to be associated with the affected platforms; however CPE information is not associated with mitigation actions, hence it is not fully possible to identify which resolution(s) can be applied to which asset(s).

***Rapid7 Vulnerability and Exploit DB***: Within Rapid7 Vulnerability and Exploit DB, each CVE entry contains several fields, out of which the *Solution Reference* and *Solution* ones provide mitigation information. The *Solution Reference* field provides a URL, which leads to the related page provided by the vendor, although sometimes no such page exists and therefore this field is not available. The *Solution* field provides mitigation information in hyphen–separated keywords, e.g. *mozilla–firefox–upgrade–64_0*. This field can be useful in terms of automated mitigation information extraction, at least to some extent. This is because for each vendor, it follows a vendor–suited structured format. Some examples of this:

- When the solution is provided by Microsoft the format is `msft-kb...`, followed by the KB code.
- When the solution is about SUSE Linux and upgrading a component, the format is `suse-upgrade-...` followed by the name of the component to be upgraded. It is accessible only via HTML page.

Rapid7 Vulnerability and Exploit DB does not provide CPE information, however it does include a pointer to NVD, which can be used to identify related CPE identifiers; CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

***Security Focus DB***: The Security Focus DB provides for each CVE entry a *Solution* tab. When an update is available, a human readable text is provided declaring that *Updates are available* and that the reader should consult the references tab or vendor advisory for more information. In the references tab, links are provided,

[68] https://tagvault.org/frequently–asked–questions–about–swids/

with relevant titles, but it's not structured, hence not automatable. In other words, the only information that can be extracted in an automated way, is if an update is available. It is accessible only through HTML page.

Security Focus DB does not provide CPE information; Security Focus DB entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

***Exploit DB***: The Exploit DB does not provide mitigation information.

***AusCert Security Bulletins***: Within the AusCert Security Bulletins database, each CVE entry contains several fields, out of which the *Remediation/Fixes*, *Workarounds and Mitigations*, *Patch Instructions*, *Resolution*, *Workarounds*, *Security Advisory Recommended Actions* and *Mitigation* ones seem to be available for obtaining mitigation information. Except the fact that there are a lot of variations in the titles as mentioned above, the information is presented in human–readable format and doesn't seem to be suitable for automated extraction. However, in some cases, namely in the *Patch instructions* and *Resolution* fields, the actual commands for applying the patch/resolution are provided, divided by version of software. Although the structure is not ideal; an automated solution could be implemented. It is accessible through HTML and RSS feed.

AusCert Security Bulletins does not provide CPE information; AusCert Security Bulletins DB entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

***CERT Vulnerability Notes DB***: Within the CERT Vulnerability Notes DB, each CVE entry contains several fields, out of which the *Solution* field provides mitigation information. This field is written in human–readable format, so it doesn't seem to offer an automated extraction–suitable structure. It is available through HTML and RSS feed.

CERT Vulnerability Notes DB does not provide CPE information; CERT Vulnerability Notes DB entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

***Common Vulnerabilities & Exposures***: The Common Vulnerabilities & Exposures database doesn't provide mitigation information.

***ICS–CERT Advisories***: Within the ICS–CERT Advisories database, each CVE entry contains several fields, out of which the *Mitigations* field contains mitigation information. The information is available in human–readable format, and thus doesn't provide an automated extraction–suitable structure. It is available through HTML and RSS feed.

ICS–CERT Advisories does not provide CPE information; ICS–CERT Advisories entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

***Japan Vulnerability Notes (JVN)***: Within the Japan Vulnerability Notes, each CVE entry contains several fields, out of which the *Solution* and *Vendor Status* ones provide mitigation information. The *Solution* field provides a clear description e.g. `Update...` followed by what must be updated, or `Use the latest installer`, which can be automated in some level. However, when the solution is *Apply Workarounds*, the workarounds are provided in human–readable format, and thus cannot be automated. The JVN is available through HTML, RSS feed.

Japan Vulnerability Notes does not provide CPE information; Japan Vulnerability Notes entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

***JVN iPedia***: Regarding the content, the remarks listed above for the Japan Vulnerability Notes (JVN) apply for JVN iPedia as well. Regarding the content access methods, JVN iPedia is additionally available in VULDEF (XML–based) format and an API is also provided.

***JC3 Bulletin Archive***: The JC3 Bulletin Archive provides several fields for each CVE entry, including *Solution*, which contains mitigation information. The information is provided in human–readable format with a link to the vendor's related page. Sometimes the link is for the actual update that needs to be installed, in which case the process is automatable, but in other cases the link is not useful. It is not structured in a way that could be useful and it also seems outdated. It is available through HTML and RSS feed.

JC3 Bulletin Archive does not provide CPE information or CVE identifiers, hence the resolution information therein cannot be directly associated with assets to which they may be applied.

***NCSC–FI Vulnerability DB***: The NCSC–FI Vulnerability DB provides several fields for each CVE entry, including the *Remediation* and *Possible solutions and restrictive measures* ones, which contain mitigation information. The *Remediation* field provides a short answer, like `Software update patch`, which in some cases can be useful for automation, but not always. The *Possible solutions and restrictive measures* field is written in human–readable format, and thus is not suitable for automated extraction. It is accessible only through HTML page.

NCSC–FI Vulnerability DB does not provide CPE information; NCSC–FI Vulnerability DB entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. It is worth mentioning that NCSC–FI Vulnerability DB entries describe the affected assets in a high level of detail, hence textual matching techniques are bound to be highly efficient in identifying the assets affected by the vulnerability. Whether affected assets are identified through textual matching techniques or retrieved through NVD pointers, mitigation actions are not linked with specific CPEs, hence the provided mitigation actions cannot be directly associated with specific assets on which they can be applied.

***VulDB***: The VulDB provides several fields for each CVE entry, including the *Countermeasures*, which provides mitigation information. It is further analyzed in *Recommended* and *Status* fields. The *Recommended* field has a short description e.g. `Patch`, `Firewall` or `no mitigation known`. The *Status* field categorizes the recommendation provided, for example for the `Patch` value, it says `Official Fix`, for the `Firewall` value the relevant text is `Workaround`. This information can be used in an automated way, however the information it provides is very generic and constitutes only a first step towards an automated mitigation action. VulDB appears to be providing the most detailed information regarding mitigation actions among all generic vulnerability databases. It is available through HTML, RSS feed and API is provided.

VulDB provides CPE information; access to it requires registration, but even in this case only few results are returned. Full access to CPE information requires a subscription, which is available for a fee. Since VulDB entries contain CVEs, these can be extracted and be used as pointers to NVD entries to extract the full list of CPEs. Whether affected assets are identified through CPEs retrieved directly from VulDB entries or retrieved through NVD pointers, mitigation actions are not linked with specific CPEs, hence the provided mitigation actions cannot be directly associated with specific assets on which they can be applied.

***SecurityTracker***: The SecurityTracker provides several fields for each CVE entry, including the *Solution*, which provides mitigation information. This field seems well–structured in the case where a fix has been issued by the vendor. It will state that a fix has been issued by the vendor, details about the fix e.g. a version code, and a link for the relevant vendor's advisory page. It is accessible only through HTML page.

***TippingPoint Zero Day Initiative***: The TippingPoint Zero Day Initiative provides several fields for each CVE entry, including the *Additional Details* one, which provides mitigation information. The information is in human–readable format but in short answers which in most cases seem to have the same structure. For example, *"Vendor has issued an update to correct this vulnerability. More details can be found at: link"*. The previous example can serve for automation up to some level. However, there are cases that a structured format is not followed, and thus not serving automated extraction purposes. It is available through HTML and RSS feed.

TippingPoint Zero Day Initiative DB does not provide CPE information; TippingPoint Zero Day Initiative DB entries include CVEs, which can be used as pointers to NVD, through which related CPE identifiers can be retrieved. CPE identifiers retrieved in this fashion will not be associated with specific resolutions.

Finally, regarding timeliness, VulDB appears to be providing vulnerability analyses in a speedier fashion than NVD. This covers the availability of vectors, scoring, references to external sources and mitigation actions.

Table 3.19 summarizes the issues discussed above for the generic vulnerability databases.

Table 3.19. Mitigation provisions for different vulnerability databases

| | Includes mitigations? | Are mitigations distinguishable? | Includes CPE? |
|---|---|---|---|
| **Nat'l Vulnerability Database (NVD)[41]** | X | X | X |
| **Rapid7 Vulnerability & Exploit DB[42]** | X | X | –<br>Indirectly, through a structurally distinguishable reverence to NVD |
| **Security Focus DB[43]** | X | –<br>(bundled into references with no means to tell apart which references contain mitigations) | –<br>Indirectly, through inclusion of a CVE, which can be used as a pointer to NVD) |
| **Exploit DB[44]** | – | – | – |
| **AusCERT Security Bulletins[45]** | X | X<br>(not uniformly listed, automation hindered) | –<br>(references to CVEs exist, which can be used as pointers to NVD) |
| **CERT/CC Vulnerability Notes DB[46]** | X | X<br>(human readable text, not easily exploitable for automation) | –<br>(references to CVEs exist, which can be used as pointers to NVD) |
| **Common Vulnerabilities & Exposures[47]** | – | – | – |
| **ICS–CERT Advisories[48]** | X | X<br>(human readable text, not easily exploitable for automation) | –<br>(references to CVEs exist, which can be used as pointers to NVD) |
| **Japan Vulnerability Notes (JVN)[49]** | X | X<br>(human readable text, to some extent exploitable for automation) | –<br>(references to CVEs exist, which can be used as pointers to NVD) |
| **JVN iPedia [50]** | X | X<br>(human readable text, to some extent exploitable for automation) | –<br>(references to CVEs exist, which can be used as pointers to NVD) |
| **JC3 Bulletin Archive[51]** | X | X<br>(human readable text, generic links only in many cases, only | – |

| | | | |
|---|---|---|---|
| | | partially exploitable for automation) | |
| **NCSC–FI Vulnerability Database[52]** | X | X<br>(human readable text, to some extent exploitable for automation) | –<br>(references to CVEs exist, which can be used as pointers to NVD; affected assets are described in detail hence a good matching level can be achieved through processing of text) |
| **VulDB[53]** | X | X | X<br>(limited for free use; full after purchase) |
| **SecurityTracker [54]** | X | X | –<br>(references to CVEs exist, which can be used as pointers to NVD) |
| **TippingPoint Zero Day Initiative [55]** | X | X<br>(human readable text, only partially exploitable for automation) | –<br>(references to CVEs exist, which can be used as pointers to NVD) |

### 3.4.3 Generic weaknesses information sources

Vulnerabilities are owing to the existence of weaknesses either in the source or the configuration of the software. In all cases, the most appropriate solution is to modify or appropriately configure the software so as to eliminate the weaknesses, but in many cases generic solutions can be applied to eliminate or reduce the risk associated with the weaknesses. These solutions include a wide range of measures, including reduction of attack surface (e.g. limiting access to threat agents), application of external identity controls (e.g. through firewalls), deprivation of necessary antecedents for vulnerability exploitation (e.g. through disablement of execution of code located in the stack segment), blocking of malicious network packets (e.g. through deep packet inspection) and so forth. While –as noted above– these solutions are suboptimal, compared to a focused mitigation, they may be used as a risk reduction technique until some permanent/more effective remediation is available.

Currently, the software weaknesses catalogue that is predominantly used is the Common Weaknesses Enumeration (CWE)[69]. CWE entries include, among other information, a *Potential Mitigations* section, in which generic solutions on how the vulnerabilities owing to the particular weakness are listed. Each potential mitigation is tagged with a category, with available mitigation categories being:

- Architecture and Design
- Build and Compilation
- Distribution
- Documentation
- Implementation
- Installation
- Operation
- Policy

[69] https://cwe.mitre.org/

- Requirements
- System Configuration
- Testing

Out of these categories, the one that would potentially be useful for applying mitigation in the context of Cyber–Trust is *Operation*, which lists actions that can be applied on the software configuration and/or the environment in order to lower the overall risk. The *System configuration* category includes some good practices for configuring the system (applicable both immediately after installation and at any point in the operation period), whereas the *Installation* category lists some generic, installation–time procedures and practices to follow. Other categories describe actions that are not relevant to Cyber–Trust’s mitigation phase.

Both the product and vendor–oriented security advisories and the generic vulnerability databases include pointers to the CWE list and/or mention the CWE identifiers, therefore it is easy to identify the weaknesses to which each of the vulnerabilities is owing. From that point onwards, we can extract the appropriate mitigation elements and instruct accordingly the security experts.

Finally, the SWE list is directly available from its source in HTML, CSV and XML formats.

# 4. Graphical security models

The use of *graphical security models* (GrSMs) is the most common methodology adopted for the assessment and investigation of network security against cyber–attackers. These models visualize the dependencies among the system assets. Hence, they offer a clear view of the ways a cyber–attacker can launch the attacks on the various system attributes, and as a result GrSMs constitute an important tool for the security analysis and the design of an effective defense strategy. Many different GrSMs have been proposed [45, 64]. The purpose of this section is not to provide an extensive review of these models, but to present the most popular ones and highlight their pros and cons, leading to the adoption of the most suitable model (or combination of models) for the Cyber–Trust project.

## 4.1 Methodology

In this section we provide an overview of the state–of–the–art in GrSMs in order to evaluate the suitability of the existing models for Cyber–Trust project. More specifically, we will highlight the main characteristics, advantages and limitations of the GrSMs proposed in literature in section 4.2, keeping in mind the needs of Cyber–Trust, so that the GrSM that will be developed is aligned with the project's ambitions. In order to assess the suitability of the various GrSMs, we define three main criteria

- Criterion I: Cyber–Trust application areas.
- Criterion II: Interaction of the GrSM with Cyber–Trust modules and services.
- Criterion III: Scalability and generation aspects of the GrSM.

which are further detailed in the following subsections.

### 4.1.1 Criterion I: Cyber–Trust application areas

The Cyber–Trust project aims at developing a multi–level cyber–defense paradigm against a wide range of cyber–attacks. For this reason, the GrSM that will be developed should be able to capture and model situations where there are multiple attackers' goals, as well as the various mitigation actions to prevent these goals from being successfully achieved.

### 4.1.2 Criterion II: GrSM interactions with Cyber–Trust modules and services

Cyber–Trust project aims at building an intelligent, autonomous mitigation mechanism based on stochastic control approaches and *game theory* (GT); to do so, a suitable GrSM needs to be selected (or developed). The GrSM will be the structure upon which the decision–making process will take place. Hence, there is need to adopt (or design) a GrSM that allows to model both the attacks and the countermeasures (i.e. the mitigation actions), along with the (probabilistic) transitions through the system's different security states.

In addition to the above, the GrSM to be selected, will also be utilized by the component of the TMS that is responsible for conducting risk analysis; further details are given in Section 6. So, the design of the GrSM has to take these dependencies into account as well.

### 4.1.3 Criterion III: scalability and generation

The scalability of the GrSM should be taken into account for all phases of the GrSM cycle: preprocessing, generation, representation, evaluation, and modification (explained later on). Especially, in the case of a dynamic environment, the modification phase should be considered carefully. Moreover, the development process and the available tools that will be required to build the selected GrSM need to be investigated. Only 10 GrSMs have tools available (not including prototypes) and only three GrSMs (i.e., AGs, ATs and MPAGs) have commercial tools [45]. Ideally, we would prefer to take advantage of any available software tools in order to be able to generate and modify the GrSM to fulfil Cyber–Trust needs. Thus, in the process of developing our GrSM, the availability of free and open–source tools will be taken into account.

## 4.2 Graphical security models' classification

The various GrSMs can be categorized into tree–based models and graphs–based models. The basic categories of tree–based GrSMs are attack trees [133, 153], defense trees [16], attack defense trees [63], attack response trees [158], and attack countermeasure trees [124]. On the other hand, the basic classes of graph–based GrSMs are attack graphs [112], multiple prerequisite attack graph [48], Bayesian attack graphs [75], exploit dependency graphs [106], and logical attack graphs [108].

Although, both tree–based and graph–based GrSMs have attracted strong scientific interest during the past years, there is significant lack of comparison between these two types in terms of general effectiveness and performance [45, 66]. Due to the growing need for effective mitigation strategies against cyber–attacks in modern networks, recent works focus on this issue. A recent study trying to conclude on which method is more effective in dealing with cyber–attacks can be found in [66]. The basic differences between these tree–based and graph–based GrSMs are next explained. A tree–based model is used to describe a *single* attack goal, while a graph–based model can present scenarios with *multiple* attack goals; in general, a graph–based model can contain cycles. Attack trees focus on the consequence of an attack, whereas attack graphs typically focus on the attacker's activity and their interaction with the targeted system [24].

The above imply that in case there is need to capture *the attack paths*, then a graph–based model would be preferred to a tree–based one. On the other hand, if the focus is the assessment of the overall network security, where only the most critical vulnerabilities of the system need to be analyzed, then a tree–based model would probably be more suitable. Graph–based GrSMs can be generated in polynomial complexity (*see* Section 5), but the evaluation phase has an exponential complexity to cover all set of attack paths or uses heuristic methods. Tree–based GrSMs can evaluate the security in a scalable manner with polynomial size complexity, but there is a lack of efficient generation algorithms for tree–based GrSMs [45].

### 4.2.1 Tree–based models

In this section we briefly review the basic tree–based GrSM categories and mention their basic properties. The following models are presented according to the chronological order that appeared in the literature (*see* Table 4.1) and are further detailed in the subsequent sections.

Table 4.1. Tree–based graphical security models

| Name | Reference |
|---|---|
| Attack tree (AT) | [128, 134, 133] |
| Defense tree (DT) | [16] |
| Ordered weighted averaging tree (OWAT) | [156] |
| Protection tree (PT) | [26] |
| Attack response tree (ART) | [158] |
| Attack countermeasure tree (ACT) | [124] |
| Attack defense tree (ADT) | [63] |
| Attack fault tree (AFT) | [65] |

#### *4.2.1.1 Attack tree*

Weiss' approach [153], which introduced threat logic trees as the first GrSM can be seen as the origin of numerous subsequent models. One of the most influencing and widely accepted models is the AT [128, 134, 133]. According to the AT formalism, the goal of the attack is represented as the root node of AT and each node refers to a sub–goal, with its children representing the ways to achieve that goal. Sub–goals are joined by logical gates (e.g. AND, OR) [134]. An example of an AT is illustrated in Figure 4.1.

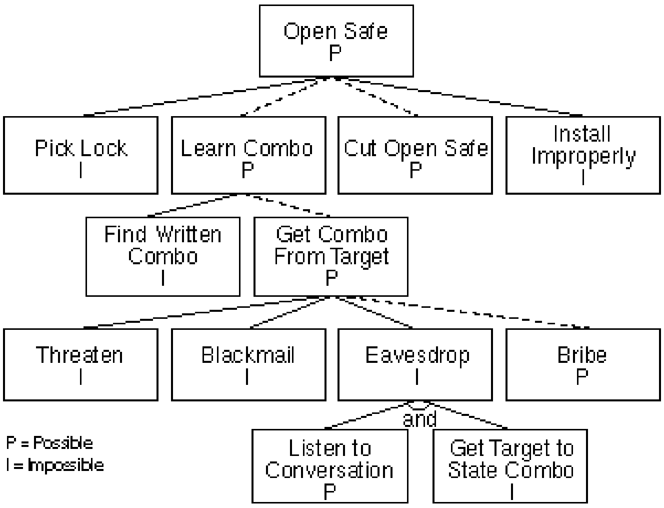


Figure 4.1. Example of an attack tree [134]

#### *4.2.1.2 Defense tree*

In 2006, Defense Trees (DTs) were introduced, which are an extension of the ATs providing the ability to model defensive actions (i.e., proactive, reactive, mitigation, remediation) along with the attack events [16]. These actions are placed at the leaf node level of DTs. Apart from enriching ATs with defensive actions, the authors use economic quantitative indexes for computing the defender's return on security investment as well as the attacker's return on attack. An example of a DT is illustrated in Figure 4.2.

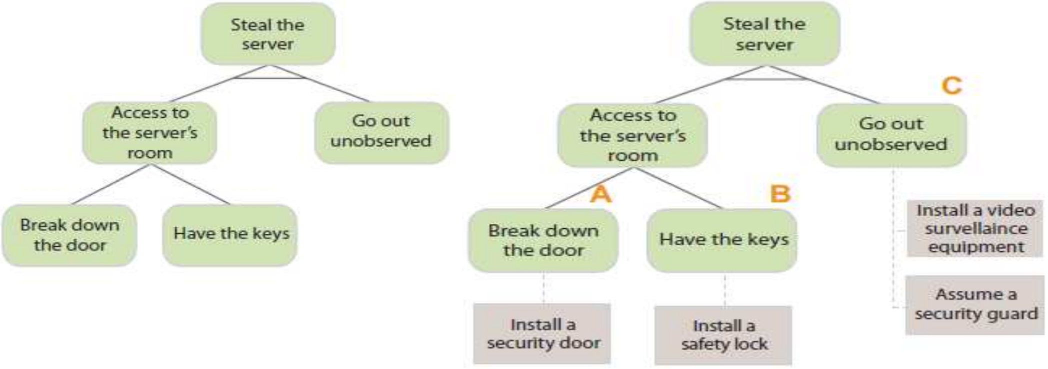


Figure 4.2. An example of an attack tree (left) and the corresponding defense tree (right) [16]

#### *4.2.1.3 Ordered weighted averaging tree*

OWAT was proposed in [156] to extend ATs in order to include partial satisfiability of logical conditions. OWATs use OWA nodes which allow the modelling of situations in which there is some probabilistic uncertainty in the number of children that need be satisfied for the parent node to be achieved, in contrast to an ''OR'' node which requires only one of the children to be satisfied or an ''AND'' node requires all the children to be satisfied. Techniques for the evaluation of an OWAT for the overall probability of success and cost of an attack are provided.

#### *4.2.1.4 Protection tree*

PTs are introduced in [26]; the nodes in PTs represent countermeasures, while in ATs nodes represent vulnerabilities. Both ATs and PTs are AND/OR trees. The root node in a PT directly corresponds with the root node in an AT, but the rest of the tree's structure may differ widely. An example of a PT is illustrated in Figure 4.3.

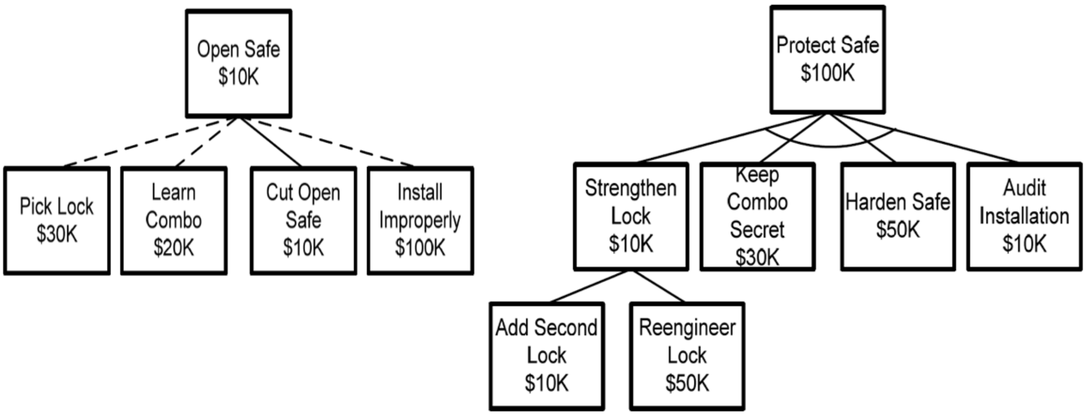


Figure 4.3. An example of an attack tree (left) and the corresponding protection tree (right) [26]

#### *4.2.1.5 Attack response tree*

In order to develop an automated intrusion response engine based on game–theoretic techniques, the authors in [158] extended ATs to the so–called ARTs. ARTs provide a formal way to describe system security based on possible intrusion and response scenarios for the attacker and response engine, respectively. They also consider the inherent uncertainties in alerts received from the intrusion detection system (IDS), i.e. due to false positives and false negatives. Unlike the ATs that are designed according to all possible attack scenarios, ARTs are built based on the attack consequences (e.g., an SQL crash); thus, the designer doesn't need to consider all possible attack scenarios that could cause these consequences [45].

#### *4.2.1.6 Attack countermeasure tree*

ACTs were developed in [124] to extend DTs to include the placement of defense mechanisms at every node of the tree and not only at the leaf node level and incorporate the probability of attack. Compared to another similar model ARTs, the ACTs do not suffer from the problem of state–space explosion (because solution in ART is resolved by means of a partially observable stochastic game model). The authors use single and multi–objective optimization to find suitable countermeasures under different constraints. In ACT, there are three distinct classes of events:

- attack events,
- detection events, and
- mitigation events.

ACT can consist of a single attack event, or an attack event and a detection event, or an attack event and multiple detection events, or an attack event, a detection event and a mitigation event, or an attack event, multiple detection events and the corresponding mitigation events. Examples of ACTs are illustrated in **Error! Reference source not found.**.

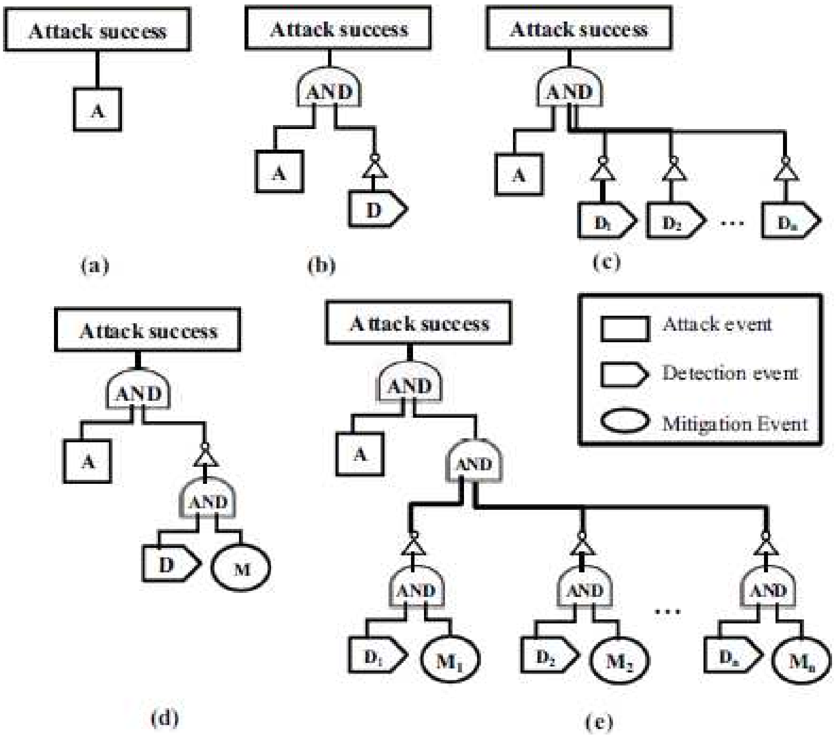


Figure 4.4. Examples of attack countermeasure trees with: (a) one attack event, (b) one attack and one detection event, (c) one attack and multiple detection events, (d) one attack, one detection and one mitigation event, (e) multiple detection and multiple mitigation

### *4.2.1.7 Attack defense tree*

In [63] ADTs are introduced and formalized, which present graphically the possible actions of the attacker as well as the available countermeasures the defender can employ. Thus, they provide a representation of the interactions between an attacker and a defender, as well as the evolution of the security mechanisms and vulnerabilities of a system. The authors in [63] develop a complete attack–defense language. In contrast to the ACT, an ADT has nodes of two opposite types:

- attack nodes, and
- defense nodes.

An example of an ADT is illustrated in Figure 4.5, where attack (resp. defense) nodes are shown in red (resp. green) color.

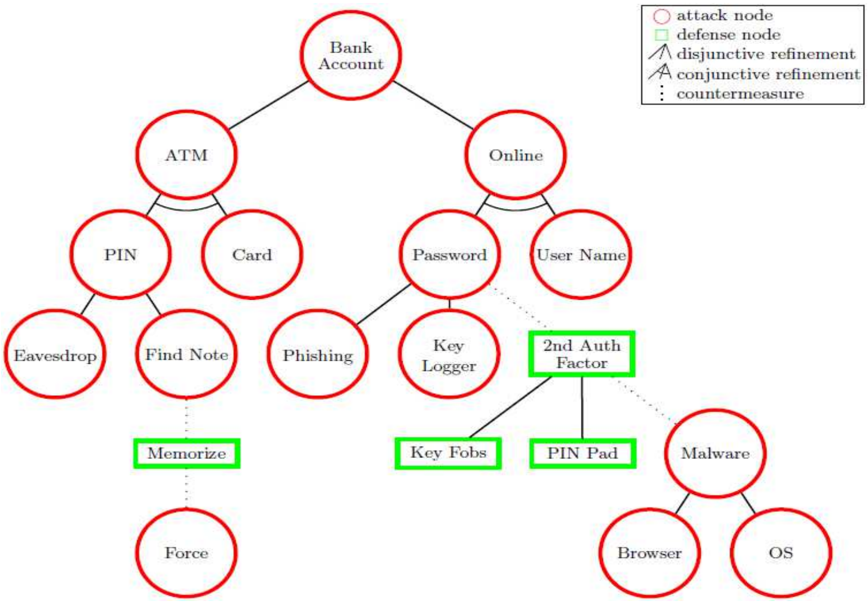


Figure 4.5. Example of an ADT for an attack on a bank account [63]

#### 4.2.1.8 *Attack fault tree*

AFTs are formalized in [65], which combine characteristics of fault trees and ATs to jointly capture the safety and security aspects. The authors equip AFTs with stochastic model checking techniques to enable a rich plethora of qualitative and quantitative analyses. AFTs model how a top–level (safety or security) goal can be refined into smaller sub–goals, until no further refinement is possible. In that case, they arrive at the leaves of the tree that model either the basic component failures, the basic attack steps or on demand instant failures. Since subtrees can be shared, AFTs are directed acyclic graphs, rather than trees. Although the underlying formalism is very similar to the AT, the widened capabilities allow the user to investigate both security and safety aspects using a single model, which other GrSMs are mostly incapable to do so.

### 4.2.2 Graph–based models

In this section we briefly review the basic graph–based GrSM categories. Likewise, the following models are presented according to the chronological order that appeared in the literature (*see* Table 4.2) and are further detailed in the subsequent sections.

Table 4.2. Graph–based graphical security models

| Name | Reference |
|---|---|
| Attack graph (AG) | [112] |
| Exploit dependency graph (EDG) | [106, 107, 104] |
| Bayesian attack graph (BAG) | [75] |
| Logical attack graph (LAG) | [108] |
| Multiple prerequisite attack graph (MPAG) | [48] |
| Compromise graph (CG) | [80] |
| Hierarchical attack graph (HAG) | [155] |
| Countermeasure graph (CMG) | [11] |

| | |
|---|---|
| Attack execution graph (AEG) | [72] |
| Attack scenario graph (ASG) | [5] |
| Conservative attack graph (CoAG) | [157] |
| Security argument graph (SAG) | [145] |
| Incremental flow graph (IFG) | [25] |
| Core attack graph (CAG) | [13] |

### *4.2.2.1 Attack graphs*

AGs [112] were proposed for network risk analysis of computer networks. AG represents attack states and the transitions between them. AGs can be used to identify attack paths that are most likely to succeed, or to simulate various attacks. In AGs a node represents states (e.g., host, privilege, exploit or vulnerability), and an edge is a directed transition from pre–condition to post–condition when an event of the state has been executed. Constructing AGs by–hand can be tedious, error–prone and impractical for an attack graph comprised of many nodes. Hence, automating the process ensures that the graph is

- exhaustive (contains all possible attacks), and
- succinct (contains only those network states from which the attacker can reach its goal).

Such a way of automated AG construction based on formal logical techniques (i.e. via model–checking) was proposed by Sheyner *et. al.* in [138], which receives as input a set of states and a transition relation and outputs the AG. The *monotonicity assumption* (on the attacker's behavior) is worth mentioning at this point; this was proposed in [7] to deal with the poor scalability of AG construction and present a more efficient solution of generating the AGs compared to [138]. The monotonicity assumption assumes that the attacker will not give up previously attained capabilities; under this assumption, the AG construction's complexity can be reduced from exponential to polynomial [45, 74]. An example of an AG is illustrated in Figure 4.6.

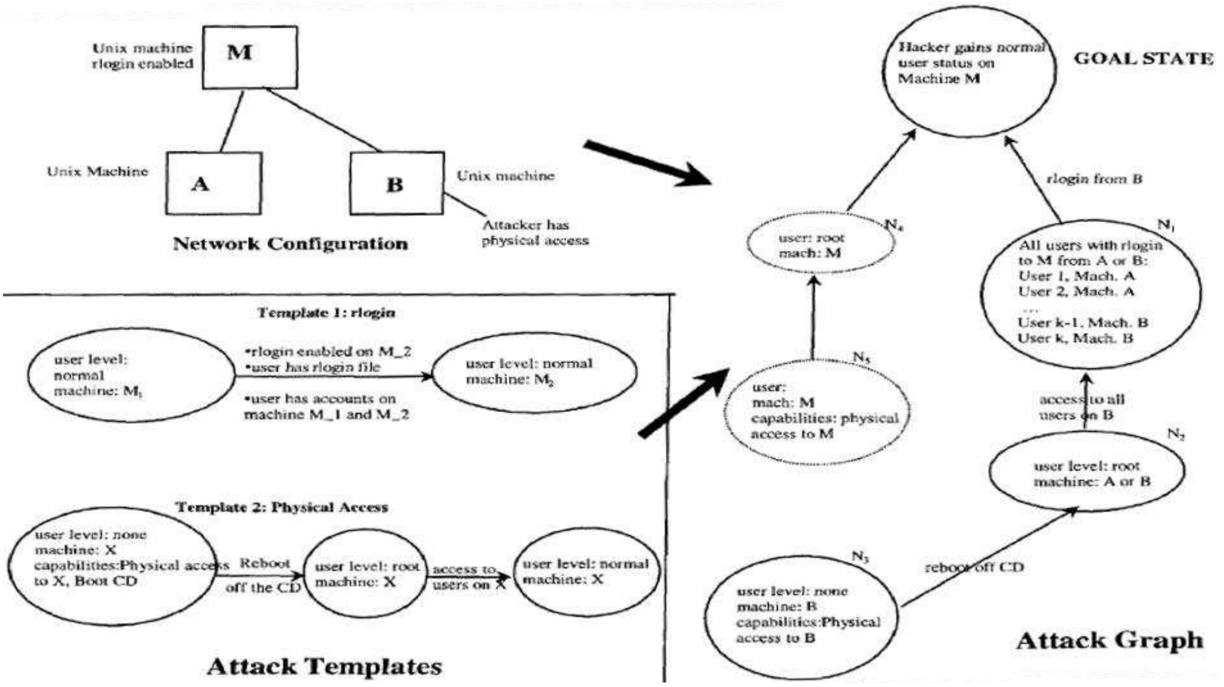

Unix machine rlogin enabled
M
Unix Machine
A
B
Unix machine
Attacker has physical access
Network Configuration
Template 1: rlogin
user level: normal machine: M_1
•rlogin enabled on M_2
•user has rlogin file
•user has accounts on machine M_1 and M_2
user level: normal machine: M_2
Template 2: Physical Access
user level: none machine: X capabilities:Physical access to X, Boot CD
Reboot off the CD
user level: root machine: X
access to users on X
user level: normal machine: X
Attack Templates
Hacker gains normal user status on Machine M
GOAL STATE
rlogin from B
N_4
user: root mach: M
N_1
All users with rlogin to M from A or B:
User 1, Mach. A
User 2, Mach. A
...
User k-1, Mach. B
User k, Mach. B
N_5
user: mach: M capabilities: physical access to M
access to all users on B
N_2
user level: root machine: A or B
N_3
reboot off CD
user level: none machine: B capabilities:Physical access to B
Attack Graph

Figure 4.6. Example of an attack graph and the generation process [112]

#### *4.2.2.2 Exploit dependency graph*

Based on the monotonic logic of attacker's behavior [7, 55], the authors in [106, 107, 104] proposed EDG. The assumption of monotonic logic also allows the resolvability of cycles and other redundancies in the dependency graph. In an EDG, the pre–conditions and post–conditions for exploits are encoded as graph nodes and edges. The resolution of cycles is part of a more general resolution of postcondition redundancies. That is, there is no reason to cycle among exploits if their postconditions remain true after an initial exploit execution, neither is there reason to execute exploits whose postconditions have already been met. As the authors state, cycles and other redundancies are common in real networks and they are violations of monotonicity that must be resolved. Indeed, in the real world, attackers themselves would avoid such redundancies. We note that in [56, 102], the authors utilized dependency graph, a structure similar to EDG, developed the *topological vulnerability analysis* (TVA) tool, which builds a dependency graph, which is a structure similar to EDG.

#### *4.2.2.3 Bayesian attack graph*

The authors in [75] proposed BAGs in order to provide a GrSM for convenient probabilistic analysis. A Bayesian attack graph can be seen as a *directed acyclic graph* (DAG) over nodes representing random variables and edges signifying conditional dependencies between pairs of nodes. The *bucket elimination* algorithm is used for belief updating and the *maximum probability explanation* algorithm is utilized to compute an optimal subset of attack paths relative to prior knowledge on attackers and attack mechanisms. Once the BAG is created, it can be used to perform probabilistic inference. The structure of the BAG does not differ from the structure of the typical AG, but the AG is treated as a Bayesian network with probabilistic assignments. Hence, the complexity and functionalities depend on the AG [45].

It should be noted though that, in a typical scenario of a BAG, each node in the graph represents a specific host of the network with a potential security violation state; two nodes may represent the same host but with different states, for instance, one with user privilege, and one with root privilege [75]. Therefore, a BAG is somehow a host–based attack graph, which is something different from the majority of the other classes of attack graphs that are being considered as state–based attack graphs.

#### *4.2.2.4 Logical attack graph*

In [108], a new approach for representing and generating AGs is proposed, referred to as LAGs, in order to deal with the scalability issues arising in model–checking approaches such as those described in [138] when applied to moderate sized networks. A LAG directly illustrates logical dependencies among attack goals and configuration information. In a LAG a node in the graph is a logical statement, which does not encode the entire state of the network, but only some aspect of it. The edges in a LAG specify the causality relations between network configurations and an attacker's potential privileges. As the authors state, Sheyner's AG [138] illustrates snapshots of attack steps, or "how the attack can happen", whereas a LAG illustrates causes of the attacks, or "why the attack can happen".

These causality relations between system configuration information and an attacker's potential privileges constitute a significant advantage of LAGs. There are two kinds of nodes in a LAG, namely

- a derivation node, and
- a fact node.

Fact nodes are further divided into primitive nodes and derivative nodes. Primitive nodes do not require a pre–condition, whereas derivative nodes require. A fact node is labeled with a logical statement and it is dependent on one or more derivation nodes, which represent a successful application of an interaction rule, where all its preconditions are satisfied by its children. The derivation nodes serve as a medium between a fact and its reasons (i.e., how the fact becomes true).

The size of a logical attack graph is polynomial in the size of the network, whereas in the worst case an AG's size could be exponential. The LAG generation tool proposed in [108] builds upon MulVAL [109], a network security analyzer based on logical programming.

#### *4.2.2.5 Multiple prerequisite attack graph*

In [48], MPAGs are introduced along with the corresponding MPAG generation tool, called NetSPA. This structure models attacker privileges and reachability conditions as state nodes in the attack graph. More precisely, the nodes in a MPAG belong to three types, namely state nodes, prerequisite nodes and vulnerability instance nodes. State nodes represent an attacker's level of access on a host and outbound edges from state nodes point to the prerequisites they can provide to an attacker. Prerequisite nodes represent either a reachability group or a credential. Outbound edges from prerequisite nodes point to the vulnerability instances that require the prerequisite for successful exploitation. Vulnerability instance nodes represent a vulnerability on a specific port. Outbound edges from vulnerability instance nodes point to the single state that the attacker can reach by exploiting the vulnerability. An example of an MPAG is illustrated in Figure 4.7.

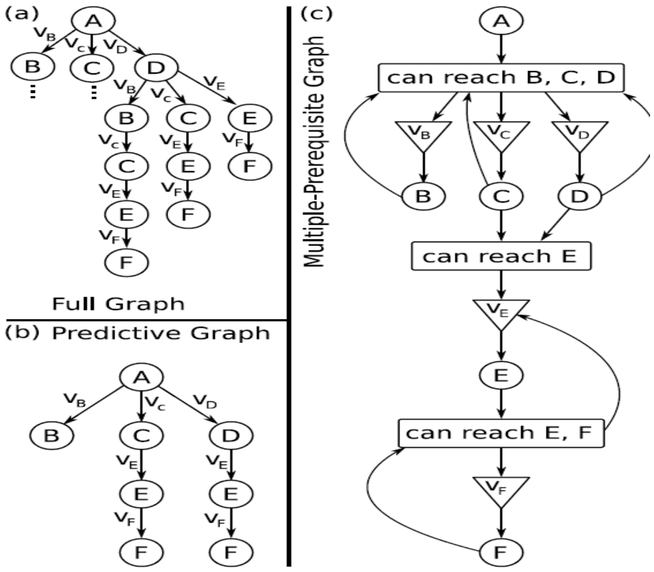


Figure 4.7. Example of (a) full graph, (b) predictive graph and (c) multiple–prerequisite graph [48]

#### *4.2.2.6 Compromise graph*

In [80], CGs were introduced to provide a quantitative measure of risk reduction. CG is a directed graph, whose nodes represent stages of a potential attack and edges represent the expected time–to–compromise for several attacker skill levels. CG provides a uniform assessment mechanism that can be applied to the evaluation of security measures in other control systems. It provides a quantitative assessment of relative time for an attacker to generate an undesired consequence. However, the CG only consists of attack states, the model lacks features to capture pre and post–conditions (i.e., vulnerabilities) [45].

#### *4.2.2.7 Hierarchical attack graph*

In [155], a novel approach was introduced to generate AGs that are suitable for large–scale networks. In a HAG two–layer AG is constructed, where the upper layer is a hosts' access graph and the lower layer is composed of some host–pair AGs. More specifically, in this two–layer model, the lower level describes all of the detailed attack scenarios between each host–pair, and the upper layer skips such detail information to show the direct network access relationships between each host–pair. An advantage of HAG is that it does not need to generate a global complete attack graph, and thus saves the computation cost. This model also utilizes the monotonicity assumption. The other assumption that HAG is based upon is the user privilege assumption, i.e., attackers only need user access privileges at source hosts when exploiting vulnerabilities at target hosts. The generation of a HAG takes polynomial time, whose upper bound computation is O($N^2$).

We note that a hierarchical GrSM called HARM [42, 43], whose formalism can be found in [44] was proposed with two layers modeling network hosts and vulnerabilities, respectively. Then, an AG is used in both the upper and the lower layers to generate the HAG. HARM is a hybrid GrSMs that uses both graph and tree–based GrSMs. AG and AT are utilized in two different layers that modeled network topology and vulnerabilities respectively. Functionalities of the hybrid GrSMs are dependent on the model used. For example, if an AG is used in both layers of the HARM, then it can provide attack sequence information, whereas the HARM with AT in both layers cannot [45].

#### *4.2.2.8 Countermeasure graph*

In [11], CMGs were proposed as an extension to ATs. The authors extended ATs in three ways. First, they consider more complex relationships among goals, actors and attacks. For example, an attack could be executed by several actors, or an actor could pursue more than one goal. Such scenarios are captured by CMGs opposed to ATs. Secondly, they include priorities assigned to goals, actors, attacks and mitigation actions or countermeasures. Finally, they include countermeasures. The edges connect goals to actors if the actor pursues the goal, actors to attacks if the agent is likely to be able to execute the attack and attacks to countermeasures if the countermeasure can prevent the attack. An example of a CG is illustrated in Figure 4.8.

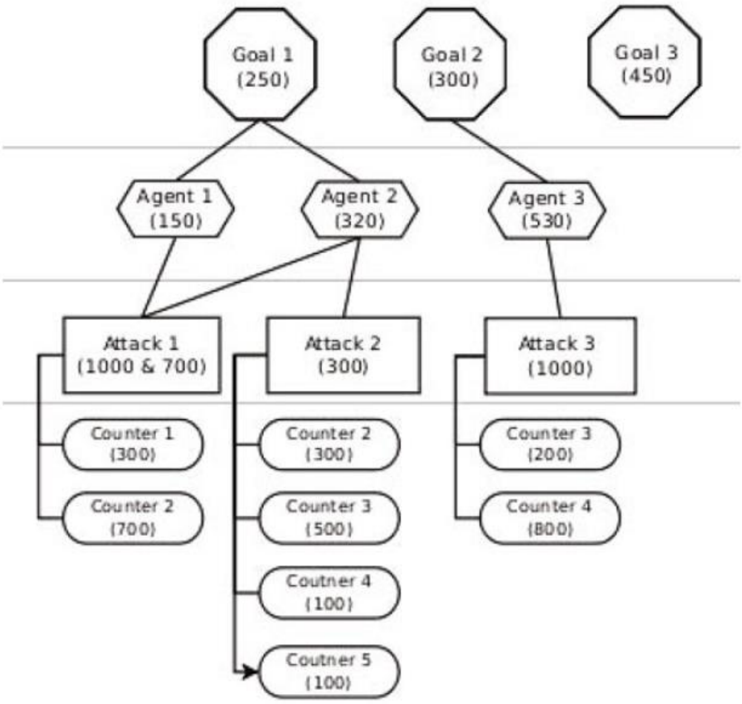


Figure 4.8. Example of a countermeasure graph [11]

#### 4.2.2.9 Attack execution graph

AEG, a similar GrSM to AG, was proposed in [72]. AEGs include adversary attack behavior models. Nodes in AEGs belong to one of the following types. *Access* nodes which describe the system–specific network domains or physical locations through which attackers can attack the system. *Skill* nodes which describe the proficiency of the attacker in executing specific types of attacks. *Attack goal* nodes, which are the attackers' target goals. *Knowledge* nodes, which are pieces of system information an attacker can utilize to achieve a goal and *attack step* nodes which are the intermediate steps of an attack. AEG has similar properties as MPAG, with an additional intermediate step of an attack and specification of compromised data or information. However, the generation method requires manual input of attacks and attackers' information from the user [45]. An example of an AEG is illustrated in Figure 4.9.

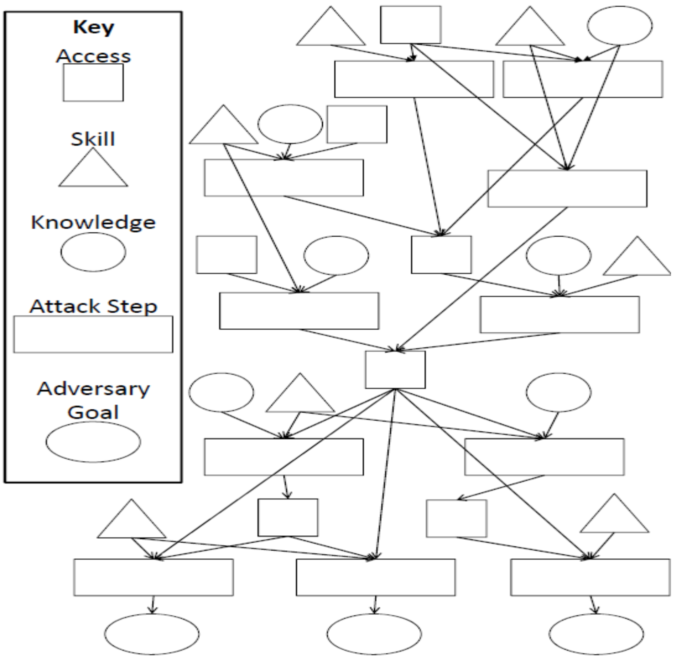


Figure 4.9. Example of an attack execution graph [72]

#### 4.2.2.10 Attack scenario graph

The combination of AGs and EDGs led to ASGs [5] towards enhancing situation awareness. In order to guarantee scalability, the authors propose efficient algorithms to track and index ongoing attacks and analyze future scenarios and show that they scale well for large graphs and large volumes of incoming alerts. Their main contributions are the following. They provide a mechanism to index alerts and recognize attacks in real–time and they provide a mechanism to integrate AG and EDG and enable real–time scenario analysis and better security decisions. More specifically, they extend AGs the notion of *timespan distribution*, which encodes probabilistic knowledge of the attacker's behavior as well as temporal constraints on the unfolding of attacks. The intuition behind ASGs is that the execution of a vulnerability (i.e., a node in AG) might cause a reduction in performance in one or more network entities (nodes in EDG). This, in turn, may affect other entities not directly affected by the exploit.

#### 4.2.2.11 Conservative attack graph

CoAGs were introduced in [157]. The authors focus on the deployment of a moving target defense system. The interesting part is that this GrSM models both gaining and losing privilege and as a result, it invalidates the monotonicity assumption [7], which is utilized by most GrSMs. An example of a CoAGs is illustrated in Figure 4.10, which is associated with the system of Figure 4.11.

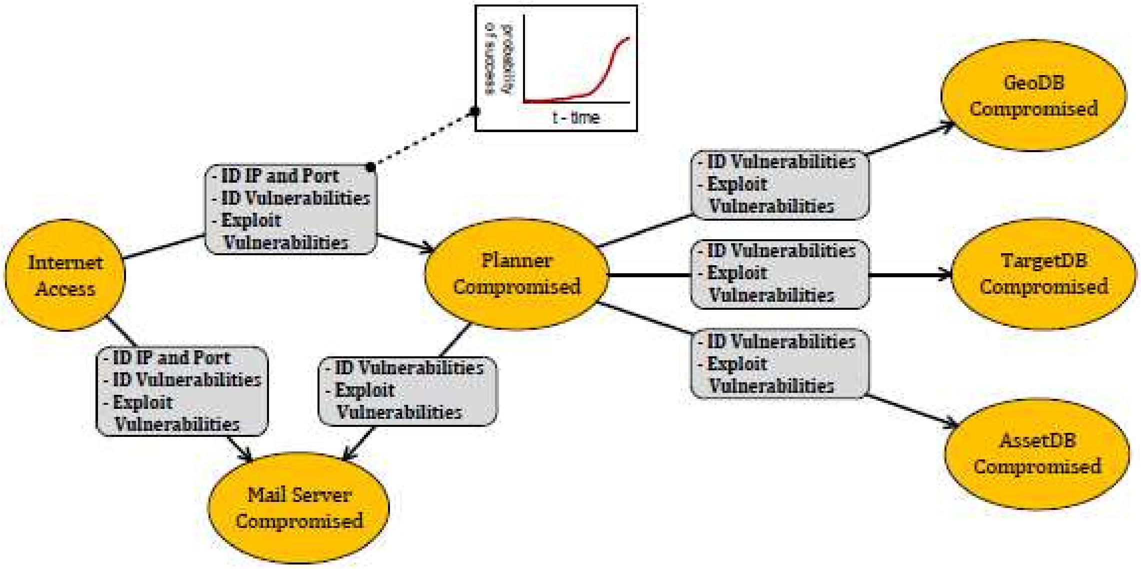


Figure 4.10. Example of a conservative attack graph [157]

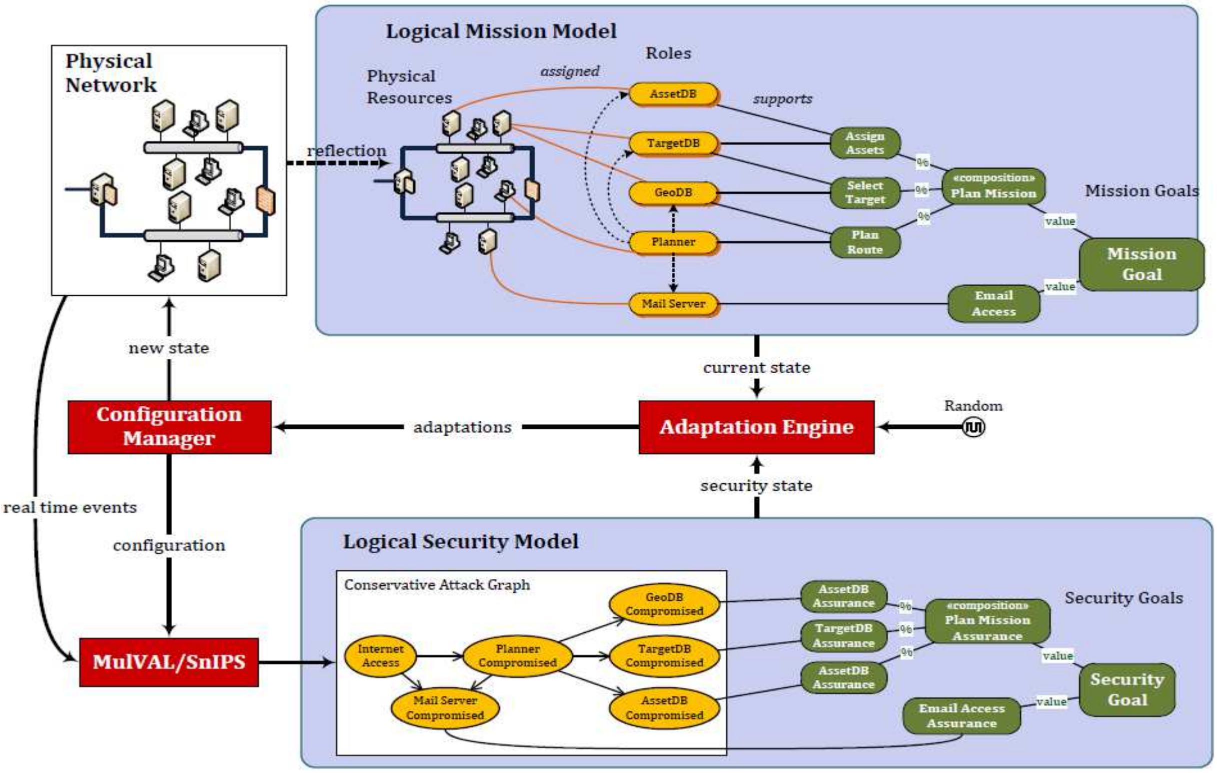


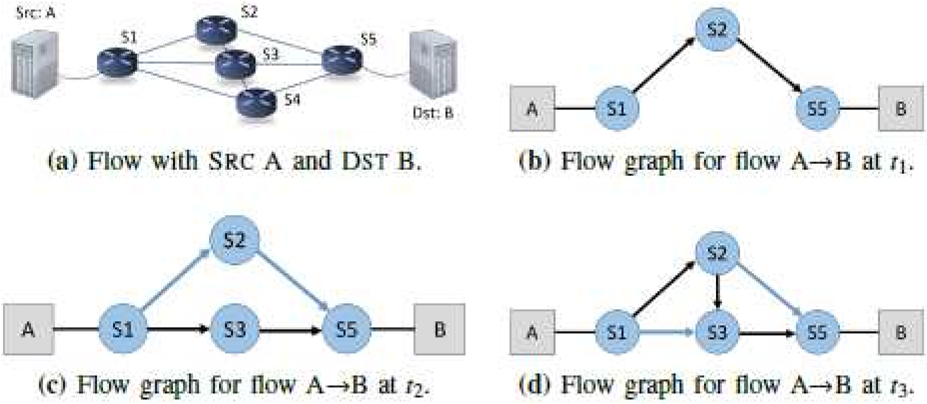
Figure 4.11. The mission planning system associated with the CoAG of Figure 4.10 [157]

#### 4.2.2.12 *Security argument graph*

A SAG is a graph whose vertices represent security goals (properties) and the edges denote dependencies between those goals. A SAG is a graphical formalism that integrates diverse inputs (including workflow information for processes executed in the system, physical network topology, and attacker models) to argue about the level of system security. They were introduced in [145] and are automatically generated by the *cyber security argument graph evaluation* (CyberSAGE) tool.

#### 4.2.2.13 *Incremental flow graph*

IFGs were proposed, along with the corresponding tool called Sphinx, in [25] for *software defined networks* (SDN). The authors aim at detecting in real–time both known and unknown attacks on network topology and data plane forwarding originating within an SDN. Sphinx incrementally builds and updates IFGs with succinct metadata for each network flow and uses both deterministic and probabilistic checks to identify deviant behavior. An example of an IFG is illustrated in Figure 4.12.



(a) Flow with SRC A and DST B.

(b) Flow graph for flow A→B at $t_1$.

(c) Flow graph for flow A→B at $t_2$.

(d) Flow graph for flow A→B at $t_3$.

Figure 4.12. Example flow and construction of the corresponding flow graph [25]

#### 4.2.2.14 *Core attack graph*

CAGs were introduced in [13] to reduce attack graph analysis complexity, handle network cycles, ease visualization aspects and support efficient subsequent analysis. Along with the formalization of CAGs, the *network attack graph generator* (Naggen) tool was developed for generating, visualizing and exploring core attack graphs. The proposed approach relies on identifying the main attack avenues towards specific network targets by performing a structural summarization process over the input network. The process essentially summarizes alternative routes between any two directly connected nodes and only keeps those routes than cannot be summarized into any other link in the graph. As a result, the obtained graphs present simpler structures which in turn can be further explored and analyzed in a hierarchical manner.

## 4.3 Comparative analysis

Due to the importance of GrSMs in cyber–security, a number of excellent survey papers are available [45, 64, 74, 61, KN46]. Perhaps the most complete survey paper in terms of comparison among the various GrSMs proposed in literature is [45]. The authors in [45] describe the usefulness of GrSMs on the basis of

- efficiency,
- application of metrics, and
- availability of tools.

The efficiency is described by the scalability and modifiability of GrSMs, which can be detailed in their phases (i.e. (i) preprocessing, (ii) generation, (iii) representation, (iv) evaluation, and (v) modification). The generation phase uses the gathered security information and generates the GrSM. The representation phase visualizes and stores the GrSM. The evaluation phase assesses the security of the networked system with given input security metrics. The modification phase captures the change in the networked system and updates the GrSM accordingly. The application of metrics distinguishes which types of security metrics can be used, and in [45] they are categorized into security–oriented (e.g., risk analysis), mathematical (e.g., a probability of an attack success), or financial impact (e.g., return on investment). The availability of tools describes how the user may access the GrSM in a form of tools [45].

Tree–based GrSMs do not suffer from the state space explosion when enumerating events, as they are only dependent on the number of events modeled. Therefore, a scalable generation of tree–based GrSMs results in scalable evaluation as well. Although generating and representing GrSMs are scalable (especially for graph–based GrSMs), there are still needs for scalable evaluation and modification of GrSMs. Graph–based GrSMs can be generated in polynomial complexity, but the evaluation phase has an exponential complexity to cover all set of attack paths or uses heuristic methods. However, many heuristic methods have been proposed that address the scalability issues in the evaluation phase. Tree–based GrSMs can evaluate the security in a scalable manner with respect to polynomial size complexity, but there is a lack of efficient generation algorithms for tree–based GrSMs. As a result, there is still great need for more robust methods of graph–based GrSM evaluation and tree–based generation methods, as well as research into how to capture changes in the networked system efficiently in GrSMs [1].

Regarding the suitability of the various GrSMs for Cyber–Trust, with regards to Criterion I, the graph–based models seem to be more suitable, as they allow for multiple attacker goals to be represented and more complex dependencies among the security conditions and the exploits. However, a hybrid model where a tree–based and a graph–based GrSM co–exist should not be excluded, as it might result in better scalability results.

Table 4.3 below summarizes the arguments of the GrSM evaluation. As discussed above, criterion I necessitates the adoption of a graph–based GrSM (although a hybrid system is not excluded); as a result, tree–based models are not included in the comparison conducted in the table.

Table 4.3. Evaluation of GrSMs

| GrSM | Criterion I4I | Criterion III |
|---|---|---|
| AG | The classic AG may not be suitable due to the fact that in AG a node in the graph represents the whole security state, whereas we aim at building a GrSM where each node represents a security condition and the edges show the dependencies among these security conditions. | There is a variety of tools for generating AGs (I.e., NuSMV, RedSeal, Skybox, Cauldron, CyGraph), but none of them is free or open–source |
| EDG | The fact that offer the option to model exploits and the relations among the security states via post–conditions / pre–conditions provide a quite suitable framework for modelling both the attacker's and defenders available actions. | Although there exists a generation tool (i.e, TVA), it is neither free, nor open–source |
| BAG | The convenience that BAGs offer for probabilistic analysis makes the consideration and adoption of the techniques used in BAGs possible. | No generation tool available |
| LAG | The formalization of LAGs, where the nodes represent logical statements and the edges causality relations between network configurations and attacker's privileges, seems not to be suitable for the envisaged GrSM for Cyber–Trust. | The generation tool MulVAL is available online and open–source |
| MPAG | The representation of security state nodes and vulnerability nodes is in accordance with the GrSM we envisage for Cyber–Trust. | Although there exists a generation tool (i.e, NetSPA), it is commercial |
| CG | CGs focus on the expected time–to–compromise for several attacker skill levels and provide a quantitative assessment of relative time for an attacker to generate an undesired | No generation tool available |

| | | |
|---|---|---|
| | consequence. The CG only consists of attack states, the model lacks features to capture pre– and post–conditions (i.e., vulnerabilities) and as a result this GrSM's characteristics are not suitable. | |
| HAG | The hierarchical structure proposed by HAGs may be a useful attribute to incorporate into our GrSM. Such an approach may be beneficial in terms of the complexity of generating the GrSM, as well. | The Safelite tool, which is the generation tool for the hybrid model HARM, is neither free, nor open–source |
| CMG | The modelling of attack goals and countermeasures, as well as the modelling of multiple actors, makes CMGs an attractive GrSM for Cyber–Trust. | No generation tool available |
| AEG | AEGs focus on the representation of the knowledge required by the attacker to achieve its goals. In cyber–Trust, we want the modelling of the possible countermeasures as well, so this model is not suitable. | The generation tool (i.e., ADVISE) is available online, but not open–source |
| ASG | ASGs combine AGs with EDGs, so they are in accordance with the envisaged GrSM for Cyber–Trust. Moreover, the algorithms proposed in ASGs for efficiently tracking and indexing ongoing attacks might be useful for the online iIRS. | No generation tool available |
| CoAG | This model invalidates the monotonicity assumption, so in case we identify this characteristic useful for the needs of Cyber–Trust, then it arises as a suitable GrSM. Otherwise, other GrSMs are more suitable. | No generation tool available |
| SAG | Not suitable because of the lack of inclusion of countermeasures in the modelling. | The corresponding tool (i.e., CyberSage) requires license |
| IFG | Not suitable due to focus on deviant behavior with regards to network flows. | The generation tool Sphinx is not free |
| CAG | The summarization process of the alternative routes between any two directly connected nodes seems to be not suitable for the iIRS model, which ideally would like to capture all available attacker and defender options. | The generation tool Naggen is not free |

Regarding criterion II the two main features that we require is the ability to model the attack and mitigation actions for the needs of the *intelligent intrusion response system* (iIRS), as documented in deliverable D2.3, and the ability to efficiently perform probabilistic inference mainly for the risk analysis task performed by the TMS. The envisaged automated defender and rational attacker formulation of the project needs a representation of all the available defender's and attacker's actions. Thus, for fulfilling the needs of the interaction between the GrSM and the iIRS, the characteristics of EDG, MPAG, CMG and ASG are suitable and we regard these GrSMs as the basis upon which our GrSM will be developed in WP5. Moreover, we aim at incorporating characteristics of BAGs into our GrSM, which are suitable for the risk analysis task.

Finally, criterion III refers to the technical issues of developing the GrSM. In this process, the possible exploitation of the suitable already available tools should be considered. Unfortunately, as it can be deducted from Table 4.3, there are no (well–established) open–source and freely available tools for the GrSMs we aim to utilize (*see* criterion II discussion in the previous paragraph). However, with respect to the scalability issues, we may incorporate ideas and the hierarchical structure from HAGs and the hybrid model HARM (uses both graph–based and tree–based GrSM) for our GrSM.

As a conclusion to the preceding state–of–the–art review and comparative analysis, the GrSM that will be developed and utilized for the needs of the Cyber–Trust project will have two main characteristics. First, it will inherit the modelling of security attributes and countermeasures in an inter–dependency fashion (GrSM that are closely related with these characteristics and structure are EDG, MPAG, CMG, ASG). The second modelling feature that our GrSM will inherit is the probabilistic inference techniques provided by BAGs. The aforementioned GrSMs are collectively in terms of the three criteria the most well suited for the objectives of the Cyber–Trust. In particular they efficiently incorporate more complex attack progressions through a hypergraph representation that allows for the sequential infiltration of the network, they are in good alignment with the information available to the attacker and defender provided by the intrusion detection system and sources of information leakage, they allow for a rigorous and detailed formulation of present and future rewards as security metrics, they are amenable to both experimental simulations and theoretical analysis through the use of stochastic games and partially observed Markov decision processes. Finally, the hierarchical structure presented in HAGs and HARM will be considered for a possible inclusion in our GrSM, because of the potential benefits in terms of scalability of the GrSM construction and modification.

# 5. Attack graph generation

As it is shown in Chapter 4, attack graphs constitute a main instrument to represent and analyze security attacks. Therefore, generating attack graphs is essential towards illustrating and evaluating the possible attack paths in networks. To this end, there are several attack graph generation techniques, whilst there are also several tools that can be used to automatically apply these techniques to produce (and visualize) attack graphs. Each of these tools is generally uniquely associated with a specific type of attack graph, i.e. with a specific security model. According to the classification presented in [59], four main issues need to be investigated towards attack graph generation:

i) Reachability analysis, which provides reachability information regarding how an attacker can reach a target.

ii) Attack template determination, which allows for deriving the relationships between a set of privileges and a vulnerability exploit. An attack template specifies the conditions required by an attacker to perform specific attacks successfully; it also describes the conditions gained by the attacker, in case of a successful attack. The attack templates form *the attack model.* The attack models can be also classified as follows [4]:

   - Prerequisite/Postcondition (Requires/Results–In) models, that is models based on prerequisites defined as the conditions needed to exploit the vulnerabilities, as well as on postcondition determined as the capabilities obtained by the attackers once the prerequisites are in place.
   - Artificial Intelligence Based models, that is models in which information of system configuration and vulnerability description is being fed as input, resulting in an attack graph according to a reasoning engine that appropriately correlates the input data.

   The vast majority of the tools follow the Prerequisite/Postcondition model (*see* also Section 3.3.1).

iii) Attack graph structure determination, i.e. determining of a proper type of attack graph.

iv) Attack graph core building mechanism, which rests with the algorithms employed to build a graph. In this context, there are logic–based methods in cases that the attack paths are created using logic deduction methods, as well as graph–based methods if the building problem is seen as a graph traversal problem and attack paths are created through graph search. Possible attack path pruning may also be decided during the core building mechanism.

In this chapter we shall provide an overview of the main currently available tools, performing a comparative study with respect to the aforementioned criteria, with the ultimate goal to reveal the appropriate tool(s) for efficiently modelling the attackers in the framework of the Cyber–Trust system.

## 5.1 Tools for generating attack graphs

In this section, we briefly review the most important tools for generating attack graphs via presenting their main characteristics. Our ultimate goal is to provide a comparative study of these tools, towards deciding which is the one that fits well with the Cyber–Trust system. For a more comprehensive survey, we refer to [59] and [45].

### 5.1.1 TVA

The *topological vulnerability analysis* (TVA) tool utilizes a database of exploit conditions, i.e. the conditions needed for exploiting vulnerabilities, as well as of postconditions that are related with the corresponding exploitations [121, 56]. By these means, combinations of possible attack scenarios can be modelled, based on the network connectivity and the corresponding privileges that the attacker acquires, according to the exploitations. Therefore, attack paths (sequences of exploits), leading to specific network targets, can be discovered.

More precisely, the underlying idea is the usage of an (exploit) dependency graph (*see* Section 4.2.2.2) to represent the preconditions and postconditions regarding an exploit. Subsequently, a graph search algorithm is used to correlate the individual vulnerabilities in a chaining mode. The TVA can be used in an off–line network security analysis, to determine optimal locations for the firewalls and intrusion detection and prevention systems [59], as shown in Figure 5.1.

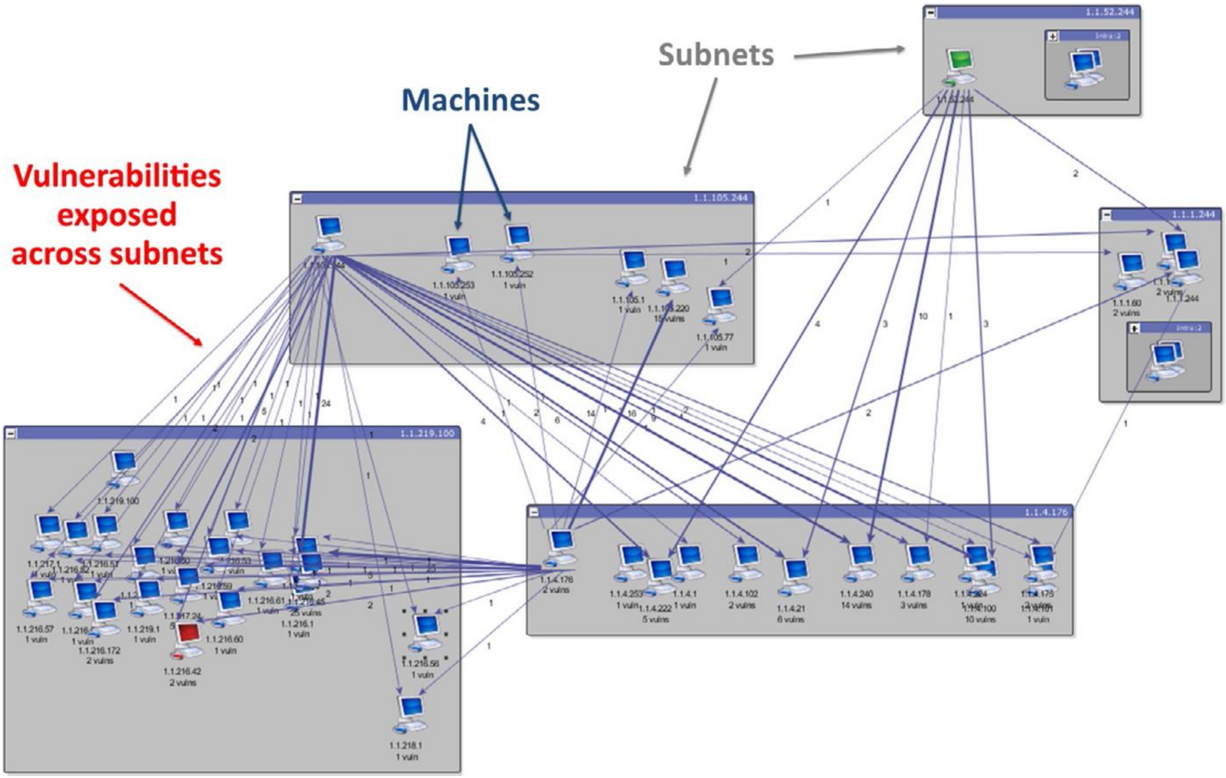


Figure 5.1. TVA attack graph visualization

The developers of the TVA tool first integrated the Nessus vulnerability scanner to automate the network discovery process. As stated in [56], each vulnerability reported by Nessus is being cross–referenced against a list of known exploits, whilst Nessus–based exploits may also have preconditions and/or postconditions for access type and privilege level. Such preconditions and postconditions are manually generated from the vulnerability information, which is available in natural language [4]. Therefore, as new vulnerabilities become known, a manual update of the conditions database needs to take place, thus raising concerns regarding the efficiency and scalability of this approach – although, in [102] and [53], an extension of the TVA is described with scalable generation algorithm. These recent versions of the TVA tool utilize the reachability concepts introduced in [48], which rest with employing the rules in firewalls, as well as the signatures in intrusion prevention systems, as an additional source of information to build a reachability matrix; moreover, trust relationships amongst the target network hosts, in conjunction with the usage relationships amongst the applications, are also used for reachability purposes [59]. Other scanner tools, such as Retina, FoundScan and Symantec Discovery are also employed [102]. The TVA tool utilizes the public text databases NVD and CVE to produce the exploitation logic. The approach of the TVA assumes the monotonicity property of attacks and it has polynomial (quadratic) time complexity [45].

Finally, it should be stressed that the TVA forms the basis of a commercial attack graph generation tool, being called Cauldron [54].

### 5.1.2 NetSPA

The *network security planning architecture* (NetSPA) is based on the so–called attacker's state, which is a combination of the locality and effect (access level) information [48]. A first version of the NetSPA is given in [10], whilst it has been significantly changed in [48]. NetSPA identifies four access levels regarding the attacker's capabilities: *root*, *user*, *DoS* and *other*. A state may provide the attacker zero or more credentials (which is defined as any information relevant to access control, such as password), whilst the locality is strongly related with the reachability – which in turn depends on whether the access level of the attacker is *root* or *user* (more generally, the reachability indicates whether a given host is able to connect to open ports on all hosts in the network [48]). Such information, in conjunction with vulnerability information from several sources, generate preconditions and postconditions. The authors in [48] refer to Nessus vulnerability scanner, the Sidewinder and Checkpoint firewalls, the CVE dictionary, and the NVD vulnerability database as the available sources of information that can be employed; the main pieces of information are network topology, vulnerability information, and credentials. In the NetSPA, reachability conditions are used to reduce the space and time complexity of building a graph [59]. The NetSPA also assumes monotonicity.

The NetSPA tool is based on the so–called multiple–prerequisite attack graphs, whose construction seems to be faster than others. The preconditions and postconditions are being produced via a logistic regression model. However, as it is stated in [4], the adopted privilege classification scheme in the NetSPA does not cover application level privileges. In the typical case, the complexity of the NetSPA scales as *O*(nlogn) in relation with the number n of hosts. A successor of NetSPA, being called GARNET [154], is also based on MPAGs, which provides a simplified view of critical steps that can be taken by an attacker, whilst it allows users to perform *what–if* experiments including adding new zero–day attacks.

A more recent version of the NetSPA is introduced in [47], which processes the rules in personal and proxy firewalls and the signatures in intrusion prevention systems to construct the reachability conditions (as described above, these principles have been also followed in the new versions of the TVA). Moreover, similarly to the TVA, trust relationships amongst the target network hosts, in conjunction with the usage relationships amongst the applications, are also used for reachability purposes [59]. Finally, features such as zero–day exploits, client–side attacks and countermeasures have been developed in this last version.

### 5.1.3 Mulval

The Mulval uses a reasoning system with Datalog tuples and rules, where Datalog is a syntactic subset of Prolog, towards constructing a LAG [109, 108]. This tool actually relies on an artificial intelligence–based model.

More precisely, in the context of the Mulval the output from the vulnerability scanner tools, as well as network topology information, are being expressed in Datalog, which are subsequently being fed into the reasoning engine. The reasoning engine consists of a collection of Datalog rules, based on the operating system behaviors and interactions between various components in the network. These rules are hand–coded and specify exploits such as code execution, file access, and privilege escalation. The Mulval, based on its inputs, analyzes the security risks of the software vulnerabilities in a correlated fashion and generates security alerts.

As stated in the [59], all the aforementioned rules are seemed to be evaluated simultaneously in parallel, which has impact on both time and storage complexity. Both complexity measures are on the order of the square of the number of the hosts in the network. However, according to recent experiments described in [4], Mulval produced significant rates of false positive and negatives.

### 5.1.4 Cygraph

Cygraph is a tool that is being developed by MITRE[70] [103], which combines data from numerous sources to build a unified graph representation for network infrastructure, security posture, cyber threats, and mission dependencies. It employs a multi–relational property graph formalism [101]. Cygraph leverages upon the topological vulnerability analysis.

The Cygraph actually uses the so–called property graphs, which are multi–relational graphs with vertices and edges of multiple types having arbitrary key/value attributes (properties). CyGraph relies on other tools and data sources for raw material to build its attack graphs. For example, as described in [101], the Cauldron tool for TVA builds network attack graphs (security posture) which are ingested into CyGraph. For cyber threats, CyGraph ingests data for both potential and actual threats, including from the Splunk log analysis tool, packet capture via Wireshark, the NVD, and *common attack pattern enumeration and classification* (CAPEC). For capturing mission dependencies on cyber assets, CyGraph ingests models developed through other MITRE tools.

### 5.1.5 CyberSAGE

CyberSAGE tool automatically generates a SAG, having manually as input information on the topology of the network, attacker actions and capabilities [145]. The various pieces of diverse information such as business processes, network topology and adversary information will be represented by CyberSAGE as input models. These will be used to initialize the graph generation engine. The tool provides also quantitative security metrics to support holistic security assessments of critical infrastructure systems. The corresponding algorithm suggests a polynomial time complexity of $O$(TV), where T is the number of templates and V is the number of vertices.

### 5.1.6 ADVISE

The *adversary view security evaluation* (ADVISE) tool provides a discrete–event simulation environment for producing network security metric values [68]. It is based on an attack execution graph, which is a set of paths determined by attack steps. An attack step is being considered as successful if the required skills, access conditions and knowledge items have been obtained by the attacker. Therefore, the authors in [68] describe the attacker profile, as the one holding the skills of the attacker and his initial knowledge about the target network.

The attack execution graph is used in conjunction with the defined attacker profiles to find the attack paths that could be followed by the corresponding attacker types. In fact, the ADVISE tool mimics, via simulation, the progress of the attacker inside the network as a series of attack steps according to the attacker profile. During the simulation, the tool computes values for the network security metrics; these can be state metrics (i.e. the average amount of time the target network is in a specific state) or event metrics (i.e. the average number of times an event occurs).

The attack decision function used by the ADVISE tool accounts for the cost, payoff and detection probability when determining the next attack step for the attacker [59]. The modeling formalism of ADVISE has been incorporated in the Möbius modeling simulation tool[71].

### 5.1.7 Naggen

The *network attack graph generator* (Naggen) is a recent security tool aiming at the generation and visualization of specific attack graphs, being called core graphs. As described in [13], Naggen is composed of three main building blocks:

[70] https://www.mitre.org/research/technology–transfer/technology–licensing/cygraph/
[71] https://www.mobius.illinois.edu/

- *Naggen Shell*, a command–line interface for configuring and controlling the generation process,
- *Naggen Core*, is responsible for the analysis and graph generation processes, and
- *Naggen Display*, which contains visualization mechanisms to display the generated attack graphs.

The main novelty of the Naggen seems to be the use of core graphs; these graphs are compact, allowing for a reduction in the analysis complexity. The main underlying idea of the core graphs rests with identifying the main attack paths towards specific network targets by performing a structural summarization process over the input network. By this summarization, the obtained graphs have simpler structures.

### 5.1.8 Evaluation – Discussion

Table 5.1 summarizes the main characteristics of the software tools discussed so far.

Table 5.1. Software tools for developing attack graphs

| Tool | Attack template | AG model | Building mechanism | Integrated with | Complexity | License model |
|---|---|---|---|---|---|---|
| TVA | Text processing–based attack template | EDG | Graph–based | Nessus, Retina, FindScan, NVD, CVE databases, etc. | $O(n^2)$ | Commercial |
| NetSPA | Manually defined attack template | MPAG | Graph–based | Nessus, Sidewinder, Checkpoint, NVD, CVE databases, etc. | $O(n \log n)$ | Commercial |
| Mulval | Manually defined attack template | LAG | Logic–based | OpenVAS, Nessus | $O(n^2)$ to $O(n^3)$ | Free[72] |
| ADVISE | Manually defined attack template | AEG | Graph–based | None (ADVISE is used for design decisions before the system is deployed or before network changes are implemented – i.e. it analyzes architectural–level vulnerabilities) | N/A | https://www.mobius.illinois.edu/ |
| Naggen | Manually defined attack template | CAG | Graph–based | N/A | N/A | Not publicly available[73] |
| CyberSAGE | Manually defined attack template | SAG | Graph–based | The modeling of the potential threats rests with a list of potential attack actions for different device classes and the required attacker properties to perform those actions | $O(nT)$, where $T$ = number of templates | License needed[74] |

[72] http://www.arguslab.org/software/mulval.html
[73] http://www.naggen.org/
[74] https://www.illinois.adsc.com.sg/cybersage/download.html

| | | | | | | |
|---|---|---|---|---|---|---|
| Cygraph | Manually–defined attack template | AG (in multi–relational form – property graph) | Graph–based | Nessus, Retina, Qualys, Nmap, NVD, Wireshark, etc. | N/A | Not free (communication with MITRE) |

If an attack template is being characterized as manually defined, it corresponds to a case that the template is manually formed by security experts. Otherwise, a text–processing based attack template refers to a template formed by applying text processing methods to the information contained in appropriate databases [59]. As main conclusions, we derive the following:

a) Most tools are not open source neither free; an exception being the Mulval tool, as well the Möbius modeling simulation tool.

b) Attack graph tools require input information which can be gathered through different software tools; this is not fully–automated, due to the fact that information on vulnerabilities are mainly described in natural language in public databases/sources. Hence, it is expected that this process should be (semi–)supervised by humans and, more precisely, security experts.

c) Although each tool utilizes a different graph model, all types of graphs are state–based and not host–based (that is their nodes do not correspond to elements of the network, but to a state related with the system/attacker status, in terms of whether vulnerabilities have been exploited). The only exception is Naggen that, according to the demo[75], generates host–based attack graphs.

d) All proposed models seem to have inherent complexity issues and thus, handling the scalability in an effective manner still constitutes a challenging research task.

## 5.2 Attack graphs for Cyber–Trust

In Cyber–Trust platform, it is necessary to implement both proactive and reactive measures for impeding potential attackers from mounting successful attacks. The above analysis illustrates that an attack graph possesses many advantages that allow for both modelling an attacker's behavior as well as for identifying and alleviating possible weaknesses in the system; moreover, attack graphs – as described in the sequel – constitute a powerful tool for performing static and dynamic risk assessment of networks.

Due to the heterogeneity of the devices that will be part of the Cyber–Trust ecosystem, which in turn results in special security aspects, appropriate attack graph generation models should be employed, being able to capture this complex attack surface. To this end, *probabilistic attack graphs* seem to be a proper path to address the security challenges.

The notion of probabilistic attack graphs is quite broad, including any attack graph which also has probabilities that model the likelihood of compromising each node of the graph, according to the specific information it carries. In a typical scenario, CVSS scores (see Section 6) can be used to model such probabilities – i.e. the probability of compromising a node $n$ while being at a node $m$ (that is the conditional probability $\Pr[n|m]$) can be estimated through the CVSS scores of the vulnerabilities corresponding to the node $n$ that can be exploited starting from the node $m$. Bayesian attack graphs, which are described in Section 4.2.2.3, present such desired properties. Although the initial definition of Bayesian attack graphs in [75] is quite strict with regard to the type of its nodes, the principles that rest with Bayesian attack graphs can be also applied to clustered structures of networks, thus generalizing the notion of a graph node; by these means, a Bayesian attack graph can be appropriately constructed to model the dependencies across clusters, via adding one edge from one node in each cluster to one node in each of the other clusters, provided that

[75] http://demo.naggen.org/

the DAG structure required for BNs is retained [90]. Such an approach may also efficiently alleviate scalability issues.

None of the software tools described in Section 5.1 seems to suits well with Bayesian attack graphs, whilst the vast majority of them are not freely available. Therefore, in the framework of the Cyber–Trust project, the modelling of the potential attacks will be implemented in an ad–hoc manner, via developing an appropriate probabilistic attack graph to capture the dependencies between the several parts of the network under monitoring, in relation with the possible vulnerabilities that might be exploited by the attacker. To this goal, several open source implementations of some algorithms generating attack graphs will be investigated; for instance, the following open source implementations will be examined in terms of their applicability and effectiveness:

- The Python implementation in github.com/Rhy0ThoM/Distributed–Attack–Graph–Generation is related with the method of distributed attack graph generation [60], which is based on a parallel and distributed memory–based algorithm that builds vulnerability–based attack graphs, with the aim to cope with the size explosion of the graph.
- The Python implementation in github.com/av9ash/AttackGraphAnalyzer calculates the probability of a root node being compromised, through the usage of a local NVD database to normalize base score and assign it as a vulnerability value for that particular node.
- The Python tool in github.com/cyberImperial/attack–graphs aims to help security administrators to reason about the risk posed to the various system components and to evaluate adversarial and defense strategies when signs of compromise have been found. This product seems also to be able to provide a visualization of the network, whilst the inference engine depends on Mulval.
- Python and C++ implementations of BAGs are given in github.com/lovingmage/IBAG.

It should be pointed out though that the aforementioned implementations are not tested, whereas their documentation is very limited.

# 6. Risk management and attack mitigation

Risk management and attack mitigation are important processes for the protection of IT infrastructures from advanced cyber–attacks. There exists a large number of risk assessment & management standards and methodologies, e.g. those by NIST [96, 93, 95] and the *International Standards Organization* (ISO) / *International Electrotechnical Commission* (IEC) [50, 51, 52, 49], providing concrete frameworks and guidelines for managing risks and threats. Managing security risks is quite a complex task that in a holistic approach involves many different levels [95]: (a) *organizational*, (b) *mission and business processes*, and (c) *information systems*. Our sole concern here is the last level, i.e. how to manage risks at the information systems level, particularly focusing on the needs of Cyber–Trust project.

Risk management is about dealing with security risks in a *proactive* way, i.e. to harden a system's security by eliminating its weaknesses and minimizing potential risks *before* the occurrence of security incidents; this is a continuous and iterative process. Most of the proposed frameworks consider threats and system's vulnerabilities in isolation to those existing in other infrastructure's elements and they work well in more typical setups, where the environments are more or less static; a high–level framework is covered in Section 6.1. The IoT ecosystem allows the formation of much more complex and dynamic networks, compared to the previous setup, where typical risk management frameworks are quite hard to implement in practice. The design of risk management methodologies that are able to cope with highly dynamic environments has already drawn the attention of standardization bodies, e.g. NIST [97], and constitutes an active research area. Although efforts have been made to transform traditional standards from static procedural activities to more dynamic approaches, e.g. in [98], the vast majority of the approaches rely on GrSMs (*see* Section 4) and are presented in Section 6.2.

On the other hand, attack mitigation refers to the procedures that have to be in place so that any defensive action is taken in a *reactive* way, i.e. during a security incident. The approach taken by Cyber–Trust project is to rely on the same models, that is GrSMs, in order to devise intelligent intrusion response and mitigation solutions. Therefore, Section 6.3 provides a classification of mitigation actions (both proactive and reactive) to allow for a sufficient degree of automation in the attack mitigation process along with a number of tools to be used for enforcing the selected mitigation actions.

## 6.1 Static (typical) risk management

As highlighted above, NIST has published a framework for risk management in [93], that includes three main phases: (a) risk assessment, (b) risk mitigation, and (c) evaluation and assessment. From the whole risk management process, we subsequently include only those steps that also provide input to the methods of Section 6.2, or are performed in a more dynamic fashion (to allow for comparison). In addition, steps having already being presented in the previous sections (e.g. vulnerability identification of Section 3) are excluded as well.

### 6.1.1 Risk assessment

In order to assess the overall risk linked to the identified vulnerabilities of an IT system (*see* Section 3), the computation of (a) the likelihood of a vulnerability being exploited, and (b) the impact that a successful exploitation will have on the system's operation and an organization's business, needs to be performed. The likelihood of an attack depends on the attacker's profile (*see* Section 7), the particular details of the vulnerabilities, as well as, the effectiveness of the security defenses in place. In [93], a *qualitative* rating of the likelihood has been given, whereas more contemporary techniques rely on *quantitative* methods that are built upon the CVSS standard, as shown in Section 6.2.1.

On the other hand, to conduct the impact analysis a security expert needs to weight information about a system's mission (services, processes, etc.), critical data (their value), and the data sensitivity. The impact of a security incident is commonly measured in terms of the loss or degradation of the main security goals *confidentiality*, *integrity*, and *availability* (CIA). Such a measurement is either quantitative (for tangible aspects, like loss of revenue, cost of patching, manpower required, etc.) or qualitative (for those aspects that cannot be measured in specific units) and therefore they are subjectively assigned to a particular magnitude. Table 6.1 provides indicative definitions of the qualitative categories.

Table 6.1. Magnitude of impact definitions [93]

| **Impact’s magnitude** | **Impact’s definition**<br>(Vulnerability exploitation may:) |
|---|---|
| **Low** | ▪ result in the loss of some tangible assets or resources;<br>▪ noticeably affect an organization’s mission, reputation, or interest. |
| **Medium** | ▪ result in the costly loss of tangible assets or resources;<br>▪ violate, harm, or impede an organization’s mission, reputation, or interest;<br>▪ result in human injury. |
| **High** | ▪ result in the highly costly loss of major tangible assets or resources;<br>▪ significantly violate, harm, or impede an organization’s mission, reputation, or interest;<br>▪ result in human death or serious injury. |

Apart from the likelihood of an attack exploiting a particular vulnerability, additional factors that could be taken into consideration towards computing the impact, might include the approximate cost of a *successful* exploitation as well as the way that this cost varies if the (successful) attack is carried out by threat actors of a specific profile.

In order to measure the risk, a risk–level matrix is commonly used [93], whose inputs are the attack’s likelihood and its impact, as determined above; the scoring granularity of these factors varies amongst the methodologies, but often three levels are used, namely *high*, *medium*, and *low*. The determination of the risk levels is subjective; a typical example is provided in the 3x3 matrix of Table 6.2.

Table 6.2. Traditional risk matrix for risk determination [93]

| | | **Threat impact** | | |
|---|---|---|---|---|
| | | **Low (10)** | **Medium (50)** | **High (!00)** |
| **Threat likelihood** | **Low (0.1)** | Low<br>10 X 0.1 = 1 | Low<br>50 X 0.1 = 5 | Low<br>100 X 0.1 = 10 |
| | **Medium (0.5)** | Low<br>10 X 0.5 = 5 | Medium<br>50 X 0.5 = 25 | Medium<br>100 X 0.5 = 50 |
| | **High (1.0)** | Low<br>10 X 1.0 = 10 | Medium<br>50 X 1.0 = 50 | High<br>100 X 1.0 = 100 |

Risk scale: low (1 – 10); medium (11 – 50); and high (51 – 100)

If the outcome suggests a *high risk*, there is a strong need for corrective measures. An existing system may continue to operate, but a corrective action plan must be put in place as soon as possible. If the outcome is rated as *medium risk*, then corrective actions are needed and a plan should be developed to incorporate these actions in a reasonable time period. Finally, if the outcome is described as *low risk*, then corrective actions might still be implemented if (otherwise, the risk is accepted). The corrective actions that are taken *proactively* from an organization, so as to mitigate or completely eliminate the identified risks, involve the identification of the proper controls and additional/alternative security mechanisms that are available for mitigating a risk. During the selection process, the following factors are taken into consideration [93]:

- Effectiveness of controls;
- Legislation and regulation;
- Organizational policy;
- Operational impact; and
- Safety and reliability.

The control recommendations resulting from the risk assessment process are provided as input to the risk mitigation process, during which they will be evaluated, prioritized, and implemented.

### 6.1.2 Mitigation strategy

The risk mitigation process is responsible for selecting and implementing the most appropriate controls for (ideally) *minimizing* an IT system's risk, while at the same time *minimizing* the impact on an organization's resources or mission, and *minimizing* the cost of implementing the selected controls. It is clear that this is a hard–to–solve problem (that becomes even harder in today's highly complex IT systems) and therefore the elimination of all risks is almost impossible in the vast majority of the cases. In a static risk management framework, a general procedure that can be followed for mitigating risks involves [93]:

- If a vulnerability exists, implement techniques to reduce the likelihood of being exploited.
- If a vulnerability can be exploited, apply proper security controls to minimize the risk of occurrence.
- If the attacker's cost is less than the potential gain, apply protections to increase the attack's cost (thus, decreasing the attacker's motivation).
- If the loss is high, apply technical and non–technical measures to limit the extent of the attack (thereby reducing the potential for loss).

The security controls that will be eventually deployed will be the result of a cost–benefit analysis aiming at determining if the cost of implementing the controls can be justified by the reduction in the level of risk. In more detail, this involves determining the impact of implementing (or not) the controls, estimating the total implementation costs (e.g. hardware/software, performance reduction, policy/procedure realization, personnel hiring/training, and maintenance costs), and assessing the implementation costs against system and data criticality. An estimate of the disruption potential or operational degradation that the application of new control will impose on the target system can be obtained from the NIST's *extensible configuration checklist description format* (XCCDF) specification [99], where the following values are foreseen:

- unknown (disruption not defined);
- low (little or no disruption expected);
- medium (potential for minor or short–lived disruption); and
- high (potential for serious disruption).

The risk remaining after the implementation of the controls is called *residual risk*. If the residual risk has not been reduced to an acceptable level, then the risk management cycle must be repeated until its value get lower than a predefined threshold.

## 6.2 Dynamic risk management on graphical models

It is clear from the risk management framework presented in Section 6.1 that such approaches –requiring the subjective analysis of threats and risks by security experts in many steps– face great challenges when they are applied in complex and highly dynamic environments [97]. Such challenges concern the large number of new vulnerabilities discovered each day, the ever–growing complexity of the IT infrastructures to be protected, the technical sophistication of the multistep attacks carried out by cyber–attackers in order to incrementally penetrate networks and systems, as well as, the inability of the current security defenses to detect such attacks.

### 6.2.1 Risk assessment

The information needed for assessing the overall risk linked to the identified vulnerabilities of an IT system, i.e. the likelihood of a vulnerability being exploited and a successful exploitation's impact, are measured in a quantitative manner using industry standards. The use of the *common vulnerability scoring system* (CVSS) is prevalent in this area; it provides a measure on how critical a vulnerability should be considered to be, so that risk mitigation efforts can be prioritized. CVSS three groups of metrics, also depicted in Figure 6.1: *base*, *temporal* and *environmental* metrics. The base metrics contain a set of features about the exploitability and the impact of a vulnerability; the corresponding *base score* (BS) is computed as $BS = ESC + ISC$ by means of the *exploitability sub–score* (ESC) and the *impact sub–score* (ISC).

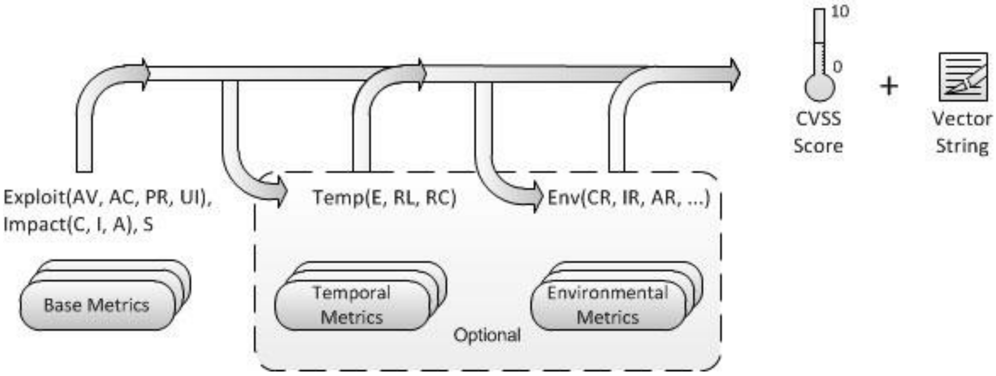


Figure 6.1. CVSS metrics and equations[76]

#### 6.2.1.1 *Setting up the scene*

Next, we present how dynamic approaches relying on GrSMs (*see* Section 4) utilize CVSS in computing the likelihood of attack, the probability of successful exploitation and an attack's impact.

**Attack likelihood.** This probability is required by all frameworks having been proposed for dynamic risk management –*see* e.g. [97, 114, 3, 76, 91, 33]. This probability can measure our *prior knowledge* about the likelihood of an attack targeting at some specific vulnerability. Clearly, the probability should depend on the availability of exploit code and the current state of exploit techniques (e.g. proof–of–concept or fully functional exploit code). This knowledge is captured by CVSS via the *exploit code maturity* (E) temporal metric that takes values in the range [0, 1]. This can also be linked to the attacker's profile (*see* more in Section 7) since the availability of easy–to–use exploit code means that even unskilled attackers will be able to launch the attack.

**Exploitation likelihood.** Given the existence of an exploit for a vulnerability, the likelihood of a successful exploitation depends on several factors. The CVSS standard provides a sufficient set of metrics on these factors, and specifically on the following

- The *attack vector* (AV) reflecting the context by which vulnerability exploitation is possible.
- The *attack complexity* (AC) describing the conditions beyond the attacker's control that must exist in order to successfully exploit the vulnerability.
- The *privileges required* (PR) documenting the level of privileges an attacker must possess before successfully exploiting the vulnerability (part of preconditions in Section 3.3).

[76] https://www.first.org/cvss/cvss–v30–specification–v1.8.pdf

- The *user interaction* (UI) capturing the need for a user to actively participate in the successful compromise of the vulnerable system.

Let $AV_{\mathrm{base}}, AC_{\mathrm{base}}, PR_{\mathrm{base}}$ and $UI_{\mathrm{base}}$ denote the base metrics corresponding to the above factors. Then, the exploitability sub–score is computed as follows

$$ESC = \begin{cases} 8{,}22\ ESC_{\mathrm{base}} & \text{if scope is unchanged} \\ 8{,}88\ ESC_{\mathrm{base}} & \text{if scope is changed} \end{cases}$$

where $ESC_{\mathrm{base}} = AV_{\mathrm{base}}\ AC_{\mathrm{base}}\ PR_{\mathrm{base}}\ UI_{\mathrm{base}}$. The expression shown above has already been adjusted by the 1,08 factor that the CVSS standard uses to weight the base score if scope is changed; so, the above is the direct contribution of the exploitability to the computation of the base score. The same expression has also been used by other works in the literature, where it is also referred to as the *probability of success* of an exploit $e_{\mathrm{i}}$ [114]; i.e. it holds $\Pr[e_{\mathrm{i}}] = 2\ ESC_{\mathrm{base}}$. Many variations of this approach can be found, e.g. by differentiating this probability for the initial and intermediate steps of a multistep attack, or even between the proactive and reactive mode of risk analysis [33].

**Impact computation.** As in the case of static risk management methods, the impact of a security incident is measured in terms of the loss of confidentiality, integrity, and availability. However, it is important that the impact measurement is *quantitative* in such dynamic framework in order to allow for immediate proactive actions or real–time reaction to ongoing cyber–attacks. Towards that direction, the CVSS standard is also used to compute the impact's rating [97]; this is accomplished by computing the impact sub–score, which is defined as

$$ISC = \begin{cases} 6{,}42\ ISC_{\mathrm{base}} & \text{if scope is unchanged} \\ 8{,}12\ (ISC_{\mathrm{base}} - 0{,}029) - 3{,}51\ (ISC_{\mathrm{base}} - 0{,}020)^{15} & \text{if scope is changed} \end{cases}$$

in CVSS 3.0, where the *scope change* flag indicates the ability for a vulnerability to impact resources beyond its means, or privileges. Likewise, the above expression equals the direct contribution of the impact to the computation of the base score. The parameter $ISC_{\mathrm{base}}$ is given by

$$ISC_{\mathrm{base}} = 1 - (1 - C_{\mathrm{base}})(1 - I_{\mathrm{base}})(1 - A_{\mathrm{base}})$$

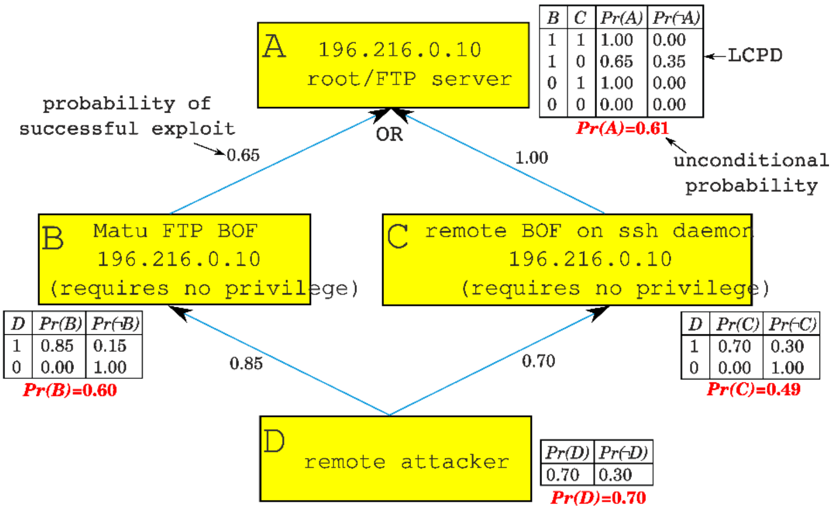
where $C_{\mathrm{base}}, I_{\mathrm{base}}$ and $A_{\mathrm{base}}$ denote the *confidentiality* impact, *integrity* impact, and the *availability* impact respectively. Note that if the terms $C_{\mathrm{base}}, I_{\mathrm{base}}$ and $A_{\mathrm{base}}$ were interpreted as probabilities, then the expression computing $ISC_{\mathrm{base}}$ above would be interpreted as the probability of admitting an impact of any form. To keep things simple, only the expressions relying on the *base metrics* are shown above. The CVSS 3.0 standard also provides modified equations due to the environmental metrics that consider the security controls available in the IT system under analysis in order to deliver more accurate set of scores. Other approaches in the literature, e.g. [33], use simpler expressions for computing the impact sub–score

$$ISC_{\mathrm{base}} = \beta_C\ C_{\mathrm{base}} + \beta_I\ I_{\mathrm{base}} + \beta_A\ A_{\mathrm{base}}$$

where $\beta_C, \beta_I$ and $\beta_A$ are weights, satisfying $\beta_C + \beta_I + \beta_A = 1$, that are related to the criticality of assets affected by a vulnerability with respect to confidentiality, integrity, and availability respectively.

#### *6.2.1.2 Dynamic risk modelling*

To deal with the drawbacks of static risk models, GrSMs in conjunction with probabilistic techniques (often based on Bayesian inference) have been proposed in order to model and assess the identified risks of IT systems [97, 114, 3, 76, 91, 33]. The nodes of attack graphs are assigned a probability that describes the likelihood of being attacked, whilst the edges of the graph are labelled with the probabilities of successful exploits (as described in the previous section). An example graph is given in Figure 6.2, where A, B, C (resp. D) are referred to as *internal* (resp. *external*) attributes of the GrSM. The probability that is given to an external attribute represents the chances of a remote attack (can be computed via the exploit code's maturity of CVSS as shown above, or it can be the security administrator's subjective belief). These models allow calculating the *local conditional probability distribution* (LCPD) at each internal attribute that represent the likelihood of being attacked given knowledge on the state of the parent node(s).



| B | C | Pr(A) | Pr(¬A) |
|---|---|---|---|
| 1 | 1 | 1.00 | 0.00 |
| 1 | 0 | 0.65 | 0.35 |
| 0 | 1 | 1.00 | 0.00 |
| 0 | 0 | 0.00 | 0.00 |

| D | Pr(B) | Pr(¬B) |
|---|---|---|
| 1 | 0.85 | 0.15 |
| 0 | 0.00 | 1.00 |

| D | Pr(C) | Pr(¬C) |
|---|---|---|
| 1 | 0.70 | 0.30 |
| 0 | 0.00 | 1.00 |

| Pr(D) | Pr(¬D) |
|---|---|
| 0.70 | 0.30 |

Figure 6.2. Example BAG illustrating probability computations [114]

To rely on Bayesian techniques for risk assessment, the BAG should be an acyclic graph; although cycles can often occur in attack graphs, due to the modeling of different attack scenarios, cycles do not increase an attack's likelihood or its impact. The dynamic aspects of this approach pertain to the ability of updating the probabilities assigned to nodes due to emerging security conditions, changes in contributing factors, or the occurrence of attack incidents. The BAG can then be used to calculate the posterior probabilities in order to re–evaluate the risk from such emerging conditions.

### 6.2.2 Mitigation strategy

The objective of dynamic risk mitigation strategies is likewise to select the security controls simultaneously minimizing the risk, the impact, and the cost of their implementation; their realization is done on GrSMs and involves solving a constrained (multi–objective) optimization problem [114, 30, 33]. Aspects concerning the cost of mitigation actions, e.g. blocking or disabling a service, patching a vulnerability, etc. are organization–specific and depend on a service's or component's criticality. The availability of mitigation actions is available from the CVSS's *remediation level* (RL) temporal metric, which may take five values: *official fix* (O), *temporary fix* (T), *workaround* (W), *unavailable* (U) and *not defined* (X) –more details about the mitigation actions are provided in Section 6.3.

Risk mitigation strategies on GrSMs that aim at *proactively* minimizing an IT system's risks are iterative in nature; this is due to the selection of some iterative solver for the optimization problem at hand or due to the implementation of a greedy algorithm for tackling efficiency. In the latter case, the steps are [33]:

- Selection of exploit node from the attack graph based on centrality measures.
- Selection of mitigation action based their cost.

At each iteration, the first step determines the exploit node to be removed from the GrSM and the second step to decide the mitigation action to be taken. This continues until the sum of the mitigation actions' cost exceeds the available security budget. In each iteration, an exploit node is removed, the graph is updated, and the new mitigation metrics are calculated; a high–level block diagram is given in Figure 6.3.

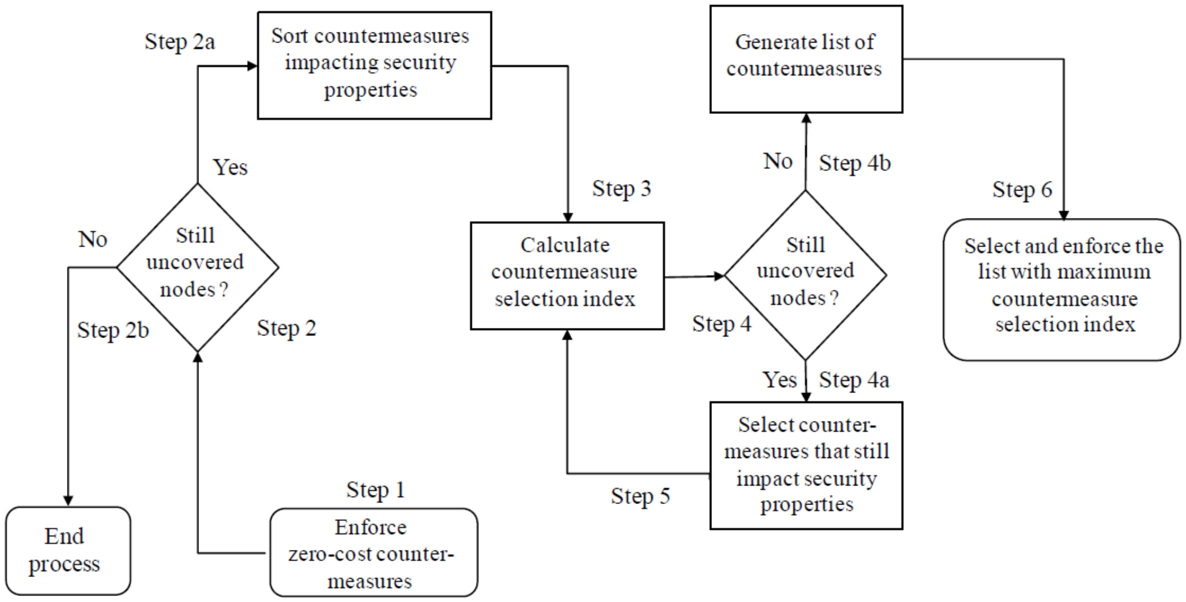

Step 2a
Sort countermeasures impacting security properties
Generate list of countermeasures
No
Step 4b
Step 6
Yes
Step 3
No
Still uncovered nodes ?
Calculate countermeasure selection index
Still uncovered nodes ?
Select and enforce the list with maximum countermeasure selection index
Step 2b
Step 2
Step 4
Yes
Step 4a
Select counter-measures that still impact security properties
Step 1
Step 5
End process
Enforce zero-cost counter-measures

Figure 6.3. Attack graph–based countermeasure selection

On the other hand, risk mitigation strategies on GrSMs that aim at operating *reactively*, select and activate new countermeasures so as to stop the propagation of ongoing attacks. On the basis of real instances of detected security violations, a priori and a posteriori steps of an attacker are mapped, and the level of risks of the GrSM nodes is updated. The set of the available countermeasures is stored in a database before the countermeasure selection process. To conduct the reactive countermeasure selection process, a number of metrics have been proposed in the literature [33], like *intrusion response cost assessment* (IRCA), *return on investment* (ROI), *return on attack* (ROA), *return on security investment* (ROSI), *return on response investment* (RORI), and *stateful RORI* (StRORI) [31, 32].

In addition to the above techniques, a number of advanced mitigation strategies have been proposed, *see* e.g. the work of [85, 86], that model the defender as an intelligent agent and rely on dynamic programming techniques for deriving the *optimal* (in the long–term) defense decisions (i.e. mitigation actions as a response to an ongoing attack), maximizing a properly designed utility function. Such approaches constitute a perfect match with the game–theoretic framework of Cyber–Trust and will be further explored in the forthcoming deliverable D5.1 that will present the state–of–the–art in this area.

## 6.3 Mitigation actions

Regardless the specific mitigation strategy having been established in the context of a static or dynamic risk management framework, the mitigation actions available to the defender need to be known in advance for dealing with the risks and threats identified during an IT system's lifetime. This is also particularly important in the design of the intelligent cyber–defense capabilities of Cyber–Trust, where the mitigation decisions will be made in an autonomous manner. Thus, in this section a classification of the mitigation actions is given (in Section 6.3.1) along with a number of available tools for enforcing the defensive decisions having been made (in Section 6.3.2).

### 6.3.1 Mitigation actions classification

Mitigation actions are typically classified as *proactive* (or *preventive*) and *reactive*. Although the needs of Cyber–Trust are primarily focusing on the latter for the efficient implementation of the iIRS, the knowledge of the former is useful for the risk assessment module of the trust management system (TMS). Since the implementation of the mitigation actions often relies on common technical controls, they are expected to share other characteristics as well, like the implementation costs, their effectiveness, etc. Thus, working with classes or taxonomies of mitigation actions, like NIST's *extensible configuration checklist description format* (XCCDF) specification [99], allows to reason about their properties in a more efficient way.

#### *6.3.1.1 High–level taxonomy*

The taxonomy of the available risk mitigation actions of Table 6.3 has been provided by NIST and is included here to facilitate the subsequent organization of an intelligent defender's available actions and also support the automated and interactive remediation.

Table 6.3. Classes of risk mitigation actions [99]

| Class | Description |
|---|---|
| Configure | Each asset stores configuration files. Among others, these files include information like functional settings that determine how the asset operates, ports that are active for operations and how they are configured, services that are enabled. The process of ensuring proper configuration involves a process of periodically checking assets against a defined configuration state which is known to be the most secure. For example, if a server allows directory listing, this will provide useful information to an attacker. |
| Combination | The combination of two approaches is a self–explanatory term. It includes cases where only one remediation technique is not enough. For example, if a host is vulnerable it might be due to insecure configuration and a missing patch for a known vulnerability, in which case both the adjustment of the configuration and the application of the patch are necessary. |
| Disable | The disablement/uninstallation of assets' components is necessary to decrease the attack surface. Usually assets come with preinstalled applications and default configurations that need to be uninstalled/disabled. Also, when under attack, the temporary disablement of a service can be crucial in a time–sensitive situation. For example, as the SSL and TLS 1.0/1.1 protocols are vulnerable, a website administrator should disable them and leave only TLS 1.2 and 1.3 enabled. |
| Enable | The need to enable/install previously disabled/missing components of an asset. It can occur when detecting a service that is disabled when it is recommended to be enabled for security reasons. It can also occur when a new component is released and its installment is recommended for security reasons. For example, when a WordPress site is vulnerable to e.g. XML–RPC attacks, there are available plugins that can be installed. |
| Patch | This involves the application of a patch, hotfix, update, etc. Patching is the process of repairing system vulnerabilities which are discovered after the components have been released on the market. A systematic checking and patch application mechanism is essential for large infrastructures. Failing to apply patches as soon as they are released leaves the assets vulnerable to attacks that can in many cases be easily deployed just by using publicly available exploit. For example, a host running MS Windows that hasn't been patched against the vulnerability used by the WannaCry ransomware can be compromised using publicly available exploit code and can result in complete host takeover. |

| | |
|---|---|
| Policy | This refers to the cases where remediation requires out–of–band adjustments to policies or procedures. Policies are sets of principles that are intended to guide actions of an organization. When a policy followed in a certain organizational procedure is found to pose a security threat, it could be necessary to be adjusted. For example, an organization that wants to provide WiFi access to clients/visitors should have a policy in place that restricts the access rights that can be obtained through this WiFi connection, e.g. by setting up an isolated guest WiFi. |
| Restrict | This includes the adjustment of permissions, access rights, filters, or other restrictions. Restrictions are placed in a network, in user accounts, and more in order to enforce access control and control the access rights and data accessibility depending on each users' credibility. For example, when detecting an employee's account as the source of an ongoing attack, the restriction of its access rights could be one possible mitigation. |
| Update | This refers to the installation, upgrade or update of the IT system. Although this has some overlap with the *patch* class, it refers to the case of installing major updates of software/hardware components of an IT system. |

In case that a particular risk mitigation action cannot be classified in one of the above classes, then it will be said to be in the *other* class (this corresponds to the class *unknown* of [99]).

#### *6.3.1.2 Proactive actions*

The use of the preventive mode is to evaluate the levels of risk that reside in the system prior to detecting attack instances. Common risk mitigation actions of this phase have been included in Table 6.4. As already mentioned above, emphasis is placed on the degree at which a mitigation action can be automated; this is reflected by specifically including such information in the action's description.

Table 6.4. Classification of proactive risk mitigation actions

| Action | Class | Description |
|---|---|---|
| System reconfiguration | Configure | Reconfiguration of an asset in order to match a configuration baseline that is known to be more secure.<br>▪ *Automation:* The secure configuration of assets can be automated in most cases on host level (e.g. servers, routers, switches, employees' machines, etc.) as there are various tools for *security configuration management* (SCM) helping reduce the manual labor.<br>▪ *Example:* If a server allows directory listing, an attacker can simply list directories, which can lead him to useful information. By using an SCM tool this would be disabled automatically. |
| System re–imaging or rebuild | Other | Wiping all the data and performing a clean install to bring a system to its default state.<br>▪ *Automation:* It can be automated on the network level using network boot options for network–based installation[77,78].<br>▪ *Example:* For an organization that provides access to its guests/clients to dedicated desktop computers, a good security |

[77] https://www.syslinux.org/wiki/index.php?title=WDSLINUX

[78] https://www.ibm.com/support/knowledgecenter/en/SS63NW_9.5.0/com.ibm.bigfix.lifecycle.doc/Lifecycle/OSD_Users_Guide/c_imaging_windows.html

| | | |
|---|---|---|
| | | practice would be setting up an automatic reimaging task for these machines. |
| System patching | Patch | Patching is the process of repairing system vulnerabilities discovered after the components have been released on the market.<br>▪ *Automation:* The detection of missing patches and their installation is a process that is automated by security management tools on host level. In many cases they will be the same tools that automate *System reconfiguration* as seen above.<br>▪ *Example:* A host running MS Windows that hasn't been patched against the vulnerability that was used by the WannaCry ransomware attack can be compromised using publicly available exploit code and can result in complete host takeover. If the Windows automatic updating option is enabled this will not be possible. |
| Software update | Update | Similar to *system patching*. |
| Deletion/ disablement of accounts | Policy | The deletion/disablement of an account when it's not being used any more as part of organizational policy.<br>▪ *Automation:* It can be simply automated on host level.<br>▪ *Example:* If an inactive account deletion/disablement policy is not in place, an employee that didn't leave in good terms with the organization might use his account to inflict damage. |
| Deletion of files | Policy | Refers to the deletion of unnecessary files that pose a threat if leaked so as to reduce such a risk.<br>▪ *Automation:* This task can be automated on host and network level (distributed storage). On host level a simple file deletion policy provided by the operating system can be used. On network level, e.g. for files that are stored in the cloud, the cloud platforms provide file deletion policies that can be set up.<br>▪ *Example:* An organization may be required to retain documents for a period of time because of compliance, legal, or other business requirements. However, if the organization keeps documents longer than required, it creates unnecessary legal risk. |
| Secure service development to prevent insider attacks | Combination (restrict/ other) | Devise secure service development methods that significantly prevent or reduce the likelihood of insider attacks.<br>▪ *Example:* It's very easy for a database administrator to become an insider threat and at some extent this happens because of insecure development from the development phase. |
| Proper configuration of access control | Combination (restrict/ configure) | Includes the proper configuration of the access given to user's accounts or guests without an account (where applicable), but also the proper configuration of the applications of which data need to be protected and network access control.<br>▪ *Automation:* The user accounts access control can be automated by user provisioning software. The application access control can be done through proper configuration as mentioned above. The |

| | | |
|---|---|---|
| | | network access control can be automated via the use of rules in a firewall, IP filter etc.[79]<br>- *Example:* If a database has not properly configured the access rights, a lower level employee could gain access to classified data. |
| Monitoring service for early detection | Other | The use of host/network–based monitoring module to examine traffic and detect attacks as early as possible.<br>- *Automation:* There are various tools available that can automate the monitoring process on every level. It can be on the network level with a NIDS, it can be on the firewall level with a *next generation firewall* (NGFW) and on the host level with an *host–based intrusion detection system* (HIDS).<br>- *Example:* If a DDoS attack is at its beginning and the NIDS detects it and reports it to the network administrator, there is a possibility of stopping the attack in its tracks. |
| Test cases to check for issues | Combination (all)/other | Deploy real–life attack scenarios in order to stress–test the systems and detect possible issues that occur.<br>- *Automation:* There could be some attack scenarios that could be carried out completely automatically from a set of hosts that would deploy attacks against the network but for more complex scenarios manual labor would be needed.<br>- *Example:* Many organizations use red team–blue team exercises to evaluate their defensive capability and harden their security. |
| Personnel education and training | Other | Provide the personnel with the knowledge required for them to apply an organization's security practices.<br>- *Example:* An organization could provide scheduled seminars to keep the securities employee up to date. |
| Search for malware | Other | Searching the hosts and the nodes of a network for malware infection.<br>- *Automation:* This process can be done both on host and network level. On host level tools like anti–virus can be used. On network level malware can be detected by monitoring traffic with traffic analysis tools e.g. Cisco ETA.<br>- *Example:* An anti–virus tool can be programmed to conduct scheduled scans and automatically remove or quarantine the malware detected. |

The actions presented in the above table are the result of best practices' analysis by considering a number of technical and academic sources; *see* e.g. [99, 100, 129, 27, 89] and the references therein.

#### *6.3.1.3 Reactive actions*

In the *reactive* mode, new countermeasures are selected and activated to stop the propagation of ongoing attacks. When real attack incidents occur, the a priori and a posteriori steps of the attacker are mapped, and the level of risks computed initially (i.e. in the *preventive* mode) are updated. Common risk mitigation actions

[79] https://searchsecurity.techtarget.com/magazineContent/How–to–use–an–automated–user–provisioning–system–for–access–control

of this phase are included in Table 6.5. Likewise, information about the degree at which an action can be automated is included in its description.

Table 6.5. Classification of reactive risk mitigation actions

| Action | Class | Description |
|---|---|---|
| Network isolation | Combination (restrict/ configure/ disable) | The isolation of a specific part of or the whole network that is under attack or infected in order to block the propagation to the rest of the network/other networks.<br>▪ *Automation:* The automation of this process can be done on an NGFW/IPS level by adding the appropriate rules.<br>▪ *Example:* Having added the appropriate rules to a an IPS like Snort, (*see* Section 6.3.2) when it detects an attack that creates a situation matching the rule, it will take the necessary actions to isolate the corresponding part of the network. |
| Affected systems isolation | Combination (restrict/ configure/ disable) | The isolation of a host/number of hosts that have been infected in order to block the propagation to further hosts on the network.<br>▪ *Automation:* If the infection is detected on the network level, then properly configured tools like NGFW/IPS can isolate the host. If the infection is detected on the host level, then there should be some sort of agent installed on the host that would alert the responsible tool to isolate the host from the network. That tool could be either host–based or network–based.<br>▪ *Example:* The anti–virus detects a malware, changes the status of the host as infected, the NGFW monitoring the network blocks inbound and outbound traffic to/from the infected host. |
| Stop a service or process | Disable | An attack can target a specific service/process, in this case stopping the service/process could stop the attack.<br>▪ *Automation:* Upon detection of the attack the defending mechanism can stop the service. This can be done for both network–based and host–based services.<br>▪ *Example:* When a DDoS attack against an Apache Server is deployed, a Web application firewall can block the Apache service on port 80. |
| Disabling of account | Restrict | Disabling an account when it's detected to be used for malicious activity or when it's under attack.<br>▪ *Automation:* This can be easily automated in case of someone trying to break in the account by proper configuration of login service. In the case of malicious activity coming from the account an alert to the administrator would be issued.<br>▪ *Example:* Malicious traffic is detected on the network, and the host origin is marked as being infected. The account currently logged in is determined and an alert is issued to the administrator. |
| Add firewall/IPS rule | Configure | Adding a rule to the Firewall/NGFW/IPS in order to block the malicious activity.<br>▪ *Automation:* It can be automated on a host and the network level through the use of Firewall/NGFW/IPS. |

| | | |
|---|---|---|
| | | ▪ *Example:* A DDoS attack is coming from a specific set of IP addresses and rules are created for blocking the inbound traffic from these IP addresses. |
| Blocking of outbound or inbound traffic | Configure | Blocking the outbound/incoming traffic associated with e.g. a specific IP address.<br>▪ *Automation:* It can be automated on host and network level with the use of Firewall/NGFW/IPS.<br>▪ *Example:* If an attack is detected that comes from a host inside the network, then a rule blocking the traffic with this host's IP address is applied. |
| Backup forensic copies | Other | Backup of forensic copies while an attack is happening before the attackers delete forensic evidence.<br>▪ *Automation:* This can be automated on network level and on host level with a scheduled task or when the network/host is marked as under attack.<br>▪ *Example:* A host is detected and marked as being compromised. The system logs are sent to an external system on the network for further process. |
| Take the system offline | Disable | In extreme cases when the damage of the attack is more massive than service unavailability, the system is taken offline to stop the attack.<br>▪ *Automation:* This can be automated on the level of alerting the administrator that this is the most cost–efficient solution, and the administrator will then allow the system to be taken offline.<br>▪ *Example:* When an attacker seems to have access to data that pose a great threat to an organization if stolen and another time–sensitive mitigation is not found, the system will be taken offline to cut access to the attacker. |
| Correlation with external organizations | Other | Receiving help or helpful information from external organizations to mitigate the attack.<br>▪ *Automation:* An alert to the other organization could be issued.<br>▪ *Example:* The same attack could target two different organizations. The second one, knowing that the other had already been targeted, can ask for information gathered on the attack so to have a more efficient defensive response. |

Likewise, the actions presented in the above table constitute part of best practices that have been proposed in the literature by a number of technical and academic sources; *see* e.g. [99, 100, 129, 27, 89] and the references therein.

### 6.3.2 Tools for enforcing mitigation

The mitigation actions presented in the previous subsections need to be enforced by the available (or new) security controls in an organization's IT infrastructure. Particularly, for the reactive mitigation actions, this process is automated at the host or network level with the use of Firewall/NGFW/IPS. Hence, in the sequel, we present a number of well–known tools that are capable of performing this step. The functionalities of the tools vary from intrusion detection/prevention (e.g. Snort, Suricata, Bro, etc.) to system hardening (e.g. Lynis, Bastille, Jshielder, etc.). In order to allow for the automated mitigation of cyber–attacks, the use of Snort or Suricata seems to be best options available.

### 6.3.2.1 *Snort*

Snort[80,81] (GPL v2.0 license) is an open–source network IPS/IDS that performs real–time traffic analysis and generates alerts when threats are detected. It can also perform protocol analysis, content searching or matching, and detect a variety of attacks and probes, such as buffer overflows, OS fingerprinting, semantic URL attacks, server message block probes, and stealth port scans. Snort can be used in three different modes of operation, namely *sniffer mode* (reads network packets and displays them on the console), *packet logger mode* (logs packets to the disk), and *network intrusion detection mode* (monitors network traffic and analyzes it against a rule set defined by the user). In the last mode, Snort performs actions, like monitoring of network traffic and analyzing against a defined rule set, performing attack classification, and invoking actions against matched rules. Useful tools for managing Snort include:

- PulledPork[82] (an open–source tool that automatically downloads the latest Snort/Suricata rules);
- Barnyard2[83] (an open–source software tool that takes Snort/Suricata output and writes it to an SQL database to reduce load on the system); and
- Snorby[84] (an open–source web–based graphical interface for viewing and clearing events logged by Snort/Suricata).

### 6.3.2.2 *Suricata*

Suricata[85] (GPL v2.0 license) is a free and open–source network threat detection engine. It works as an IDS, an IPS and *network security manager* (NSM). It utilizes externally developed rule sets to monitor network traffic and provide alerts to the system administrator when suspicious events occur. Furthermore, it provides unified output functionality and pluggable library options to accept API calls from other applications. Some further features of Suricata include[86] the ability to perform off–line analysis of PCAP files, decoding of packets and protocols, and utilize information about the reputation of IPs. It is extensible through Lua scripting and can be managed by the tools that were also presented above in Snort.

### 6.3.2.3 *Bro (aka Zeek)*

Bro[87] (BSD license) is an open–source, UNIX–based NIDS which monitors network traffic and looks for suspicious activity. It performs attack detection through *signature–based* detection methods but also through *anomaly–base*d detection methods. Furthermore, it keeps extensive logs which are really useful for forensics. Some additional features include the ability to perform offline traffic analysis, analysis of application–layer protocols (including files' contents), as well as detection and analysis of tunnels. It can use external programs and alternative backends, while it is extensible through a Turing–complete language for expressing arbitrary analysis tasks. Useful tools[88] to be used with Bro include:

- Broccoli (the Bro client communications library);
- Syslog2bro (tool to send syslog messages to Bro via Broccoli); and

[80] https://www.snort.org/

[81] https://snort–org–site.s3.amazonaws.com/

[82] https://github.com/shirkdog/pulledpork

[83] https://github.com/firnsy/barnyard2

[84] https://github.com/Snorby/snorby

[85] https://suricata–ids.org/

[86] https://suricata–ids.org/features/all–features/

[87] https://www.bro.org/

[88] https://www.bro.org/community/software.html

- Snort (integrates with Bro).

#### 6.3.2.4 *Sagan*

Sagan[89] (BSD–3–Clause license) is an open–source high performance, real–time log analysis and correlation engine with the ability of monitoring any type of device or system. It uses a Snort–like rule set for detecting malicious activities in a network. This means that the events detected can be stored to a Snort database (unified2/barnyard2) and the event will be correlated with Snort. This was done to maintain compatibility with the rule management software (pulledpork). Additionally, it is compatible with all Snort consoles, like Snorby and Sguil. It supports many different output formats, log normalization, GeoIP detection and script execution on event. It is rather a follow–up of Snort with not much additional features to offer.

#### 6.3.2.5 *Bastille*

Bastille[90] (GPL v2.0 license) is a system–hardening/lockdown program that enhances the security of a Unix host. It configures daemons, system settings and firewalls to be more secure. It is composed of a set of Perl scripts that run as an interactive program, asking questions for each step of the hardening process. For each step, an explanation is provided, to help the user understand what security measures will be applied and why, but also the option to choose whether the measures will be applied or not. Furthermore, the user's choices can be saved in a file for use in remote deployment to other machines.

#### 6.3.2.6 *CIS–CAT*

The *center for Internet security* (CIS) *configuration assessment tool* (CAT)[91] (license model is not available) compares the configuration of IT systems to CIS benchmarks and allows system administrators to ensure the security status and that it conforms to the configuration specified in the benchmark. The process performed, referred to as benchmarking, is the process of comparing the organization's activities to similar organizations' or to accepted best practices. The free version, CIS–CAT Lite, provides benchmarks for Windows 10, Ubuntu, Mac OS and Google Chrome, and also provides a GUI and HTML report export functionality.

#### 6.3.2.7 *Docker Bench for Security*

Docker Bench for Security[92] (Apache v2.0 license) is a set of Bash shell scripts that check common best practices for deploying Docker containers in a production environment. The tests are automated and useful and well–organized output is given to the user. The tests are compliant with a CIS Benchmark created for Docker[93]. Furthermore, it's open–source and free to use.

#### 6.3.2.8 *Jshielder*

Jshielder[94] (GPL v3.0 license) is an open–source automated hardening Bash script designed for Linux servers. Its aim is to help system administrators and developers to secure their Linux servers. It installs the necessary packages needed to host a web application and hardens the Linux server with little user interaction. There is also a newly added script that follows the CIS Benchmark Guidance for securing Ubuntu Linux systems.

[89] https://quadrantsec.com/sagan_log_analysis_engine/
[90] http://bastille–linux.sourceforge.net
[91] https://learn.cisecurity.org/cis–cat–landing–page
[92] https://github.com/docker/docker–bench–security
[93] https://www.cisecurity.org/benchmark/docker/
[94] https://github.com/Jsitech/JShielder

### *6.3.2.9 Lynis*

Lynis[95] (GPL v3.0 license) is an open–source tool used for auditing, system hardening, and compliance testing for UNIX–based systems. It provides insights on how well a system is hardened and what an administrator can do to enhance its security defenses. It has various uses, such as security auditing, compliance testing, penetration testing, vulnerability detection and system hardening. It is extensible through available plugins and supports many standards, including CIS Benchmarks, NIST, NSA, and OpenSCAP data.

### *6.3.2.10 Microsoft attack surface analyzer*

Microsoft *attack surface analyzer*[96] (license by Microsoft) is a tool meant primarily to understand the changes that occur in the attack surface of a Windows OS after the installation of additional software. It works by analyzing the files and registry keys that have been added or updated. More specifically, it runs before the installation of the additional software in question in order to create a baseline. After the organization of the software it runs again to analyze the changes in the attack surface based on the baseline created before.

### *6.3.2.11 Microsoft security compliance toolkit*

Microsoft *security compliance toolkit* (SCT)[97] (license by Microsoft) is a set of tools enabling administrators to compare their enterprise's *group policy objects* (GPOs) with Microsoft–recommended GPO baselines or other baselines, edit them, store them in files, and apply them. The set consists of:

- Security baselines for Windows 10, Windows Server, and Microsoft Office;
- Policy analyzer tool (analyze and compare sets of GPOs); and
- Local GPO tool (command–line utility to help automate local group policy management).

### *6.3.2.12 OpenSCAP*

The *security content automation protocol* (SCAP)[98] (LGPL v2.1 license) is a U.S. standard maintained by NIST. The OpenSCAP project is a collection of open–source tools for implementing and enforcing this standard. It includes the following tools:

- OpenSCAP base (command–line configuration and vulnerability scanning);
- OpenSCAP daemon (continuous evaluation of the infrastructure's compliance with a SCAP policy);
- SCAP workbench (custom security profile creation and remote system scanning from a desktop);
- SCAPTimony (centralized storage of scan results); and
- Atomic scan (to scan Docker containers for vulnerabilities and compliance issues).

### *6.3.2.13 Zeus*

Zeus is an *Amazon web services* (AWS)[99] (license by MIT) auditing and hardening tool. It checks the security settings according to the profiles the user creates and changes them based on the recommendations of the CIS AWS Benchmark. Identity and access management, networking, monitoring, and logging are included amongst its functionalities.

[95] https://cisofy.com/lynis/

[96] https://www.microsoft.com/en–us/download/details.aspx?id=24487

[97] https://www.microsoft.com/en–us/download/details.aspx?id=55319

[98] https://csrc.nist.gov/projects/security–content–automation–protocol

[99] https://github.com/DenizParlak/Zeus

# 7. Cyber–attackers' profiling

The term *cyber–attackers* refers to the individuals or groups targeting infrastructures, computer networks and systems along with their IoT counterparts (e.g. Mobile phones, IP cameras, smart houses, etc.). They have malicious intent that varies based on the *type* and the *motivation* of the attacker. Three categories of attackers can be identified regarding their location and knowledge regarding the target organization [87]:

- *Internal to the organization.* They are also known as *insiders*, and they have high level of knowledge about the target's network, systems, security, policies and procedures. According to the 15th annual CSI Computer Crime and Security Survey reports [20], there are two threat vectors contributing to insider threats, namely organization's employees having: (1) malicious intents (e.g. to disclose and/or sale non–public information; (2) non–malicious intents (e.g. they have made some unintentional mistake). The majority of the losses are due to the latter threat vector.
- *External to the organization.* Compared to the insider threats, such attackers have to spend a great amount of time before the attack for gathering information on the target, due to their limited prior knowledge.
- *Mixed groups.* They are comprised of both internal and external attackers.

## 7.1 Taxonomy of attackers

This section presents a taxonomy of cybercrime actors, providing information on their motives, scope, targets and level of expertise. In general, the cybercrime actors are broken down into seven categories:

- *Virus and hacking tools coders*: Individuals or teams of expert programmers, elite-hacking tool coders with excellent computer skills. The main focus of these actors is to develop and distribute malicious software (i.e. computer viruses, worms, rootkits, exploits, etc.) and hacking toolkits possibly to have a financial gain. The main *buyers* are non–expert individuals who want to become hackers (e.g. Script kiddies) [126]. They can launch and orchestrate complex attacks.
- *Black hat hackers*: Hackers (regardless whether they are black, white, or grey hat) are using almost the same tools and techniques, but with different motives and goals. In particular, black hat hackers are hackers with excellent computer skills (elite) that undergo illegal activities – other actors of this taxonomy are also characterized as black hats in the literature (e.g. hacktivists). Their primary motive is to earn money (e.g. Hacking as a Service) and in certain cases to cause significant damages (e.g. destroy/steal confidential data) [78, 126].
- *Script kiddies (SK) and cyber–punks (CP)*: These two groups have many similarities. As they are not professional hackers, they use existing tools to launch attacks due to limited technical knowledge. SK's main motives are fun, fame and adrenaline rush while CP's motives are mainly based on their ideology against the authorities, to gain fame and public recognition [122].
- *Hacktivists*: Hacktivism, one of the digital forms of activism, is employing hacking skills and tools to attack governmental institutions and private organizations. Hacktivists are working in groups that are formed by socio–political and ideological beliefs. They are acting anonymously and share their ideas aggressively using criticism instead of engaging in healthy debates [140].
- *Cyber–warfare/state–sponsored attackers*: They are sponsored and driven by countries aiming at causing damage by gaining illegal access to state and trade secrets, technology concepts, ideas and plans, and in general artefacts of high value for a country or state. They quite often target at critical infrastructures and in general they seek to damage a state's economy [118].
- *Cyber–terrorists*: Terrorist groups are increasingly using the Web to recruit and train new members, share information, and organize attacks in the real world. Furthermore, terrorist organizations using the anonymity and security of the Dark Web, disseminate training guidelines for cyberattacks, to less experienced supporters [22]. These groups will either employ or recruit Black hat hackers, due to

their ideology and beliefs, that will subsequently act on their behalf to launch cyber–attacks (e.g. United Cyber Caliphate).

- *Cyber–criminals*: It is common knowledge that criminals are using the Web to sell and transfer illicit goods and materials. For this taxonomy, the term *cyber–criminal* is used for a variety of cybercrime stakeholders in order to conduct traditional crimes through the use of computer systems (e.g. drug and firearm dealers, production and distribution of child abuse material, financial fraud, human trafficking, etc.).

The aforementioned actors can be distinguished based on their motivation, objectives, and skills. In the deliverable D2.3 (Cyber–Trust use cases) two main domains were identified, *Smart Homes* and *Mobile (cellular) Devices*, where the *Internet service provider* (ISP) and the *telecommunications operator* provide the backbone infrastructure; thus, attackers can not only target both domains, but also the infrastructure that is being provided.

Based on the aforementioned taxonomy, Cybercriminals and Hacktivists have the least motivation to target these domains and their respective infrastructures. Thus, the subsequent analysis in Sections 7.2 and 7.3 will exclude these two categories. Furthermore, the deliverable D2.1 (Threat landscape: trends and methods) introduced eight *threat categories*:

- *Network–level threats:* this includes threats pertaining to the three bottom layers of the OSI network reference model (physical, data link, and network layer). Threats for the SDN infrastructure are also included in this group.
- *Cryptography–related threats:* this group includes threats related to the lack of cryptography, the use of weak protocols and ciphers or cryptanalysis.
- *Hardware/sensor–level threats*, including threats related to the hardware or sensors and actuators. Since hardware is in many cases coupled with the firmware, some firmware attacks are included here.
- *Malware:* this group relates to software intentionally designed to cause damage to a computer, server or computer network.
- *Threats for smart grids:* this includes threats that are specific to the environment of smart grids.
- *Technical/application development–related threats:* this category includes threats that are related to the application layer.
- *Threats necessitating actions by the victim user:* this is related to attacks attempting to trick victim users to (unwillingly) cooperate to the attack (e.g. phishing).
- *Generic / miscellaneous threats:* this category contains all other threats, including policy–related threats, targeted attacks as well as threats that could not be meaningfully placed under the seven specific categories above.

To yield the attackers' profile, the involvement of threat actors in launching attacks from the above threat categories, which fit into the context of Cyber–Trust, will be discussed; as a result, the *cryptography–related threats* will be excluded from the following analysis (the same holds for the *threats necessitating actions by the victim user*, as these are mitigated by increasing awareness and intensifying security training). The threat category, *threats for smart grid* will be replaced from the *critical infrastructures* threat category in order to encapsulate all relevant infrastructures.

## 7.2 Attackers modelling and related metrics

In this section the correlation of the aforementioned taxonomy of attackers will be depicted with:

- The threat posed based their skill level [23]; this correlation will provide a mapping of the technical skills of the attackers and their involvement in the specific threat categories.

- The various attack metrics (attack vector, attack complexity, and privileges required for exploiting a vulnerability) as provided by the CVSS standard [6].

Table 7.1 provides a mapping between the aforementioned type of attackers and threat categories; it is based on their motives, objectives and skills (thus, illustrating what they would target at and by what means).

Table 7.1: Threat actors and their involvement/capability level

| | Virus and hacking tools coders | Black hat hackers | Script kiddies & cyber–punks | Cyber–warfare/state sponsored attackers | Cyber–terrorists |
|---|---|---|---|---|---|
| **Network–level attacks** | X | X | X | X | X |
| **Hardware/sensor–level threats (physical damage, etc.)** | | X | | X | X |
| **Malware** | X | X | X | X | X |
| **Critical infrastructure attacks** | X | X | | X | X |
| **Application–level attacks** | X | X | X | X | X |

X High capability level **and** primary threat
X Low capability level **or** *not* primary threat

Table 7.2 provides information on the correlation between the attackers' profile and the CVSS metrics in terms of possible exploitability and skills. The metrics that have been employed from the CVSS standard contribute in determining the likelihoods of (a) launching an attack and (b) succeeding in an attack for each type of attacker. As shown in Section 6.2, the attack likelihood is determined based on the existence of known vulnerabilities in a target system, along with the availability of known exploits (which can be classified as easy to use or complex to use); moreover, the computation of a successful exploitation likelihood depends on the attack complexity (low/high), the attack vector (network/adjacent/local/physical), as well as, the privileges required (none/low/high).

Table 7.2: CVSS metrics and attacker's profile

| | | Virus and hacking tools coders | Black hat hackers | Script kiddies & cyber–punks | Cyber–warfare/state sponsored attackers | Cyber–terrorists |
|---|---|---|---|---|---|---|
| **Info** | Vulnerability (publicly-known) existence | | | | | |
| | Yes | X | X | X | X | X |
| | No | X | X | | X | X |
| **Attac** | Exploit's (public) availability | | | | | |
| | Yes | X | X | X | X | X |

| | | | | | | |
|---|---|---|---|---|---|---|
| | No | X | X | | X | X |
| | Exploit's complexity | | | | | |
| | Easy to use | X | X | X | X | X |
| | Complex to use | X | X | | X | X |
| Exploitation likelihood | Attack vector | | | | | |
| | Network | X | X | X | X | X |
| | Adjacent | X | X | X | X | X |
| | Local | X | X | X | X | X |
| | Physical | | X | | X | X |
| | Attack complexity | | | | | |
| | Low | X | X | X | X | X |
| | High | X | X | | X | X |
| | Privileges required | | | | | |
| | None | X | X | X | X | X |
| | Low | X | X | X | X | X |
| | High | X | X | X | X | X |

X High capability of exploitation and attack
X Low capability of exploitation and attack

Table 7.3 below presents the number of known vulnerabilities categorized based on their CVSS score [23]. Even though there exist 14.961 vulnerabilities with score in the range 9–10, this doesn't mean that all these vulnerabilities are complex to exploit. By analyzing these vulnerabilities, it is evident that even SK & CP could potentially use them.

Table 7.3: Distribution of all vulnerabilities by CVSS scores

| CVSS Score | Number of vulnerabilities | Percentage | CVSS Score | Number of vulnerabilities | Percentage |
|---|---|---|---|---|---|
| 0–1 | 1.731 | 1.60% | 5–6 | 21.359 | 19.30% |
| 1–2 | 846 | 0.80% | 6–7 | 14.741 | 13.30% |
| 2–3 | 4.297 | 3.90% | 7–8 | 25.044 | 22.60% |
| 3–4 | 3.690 | 3.30% | 8–9 | 494 | 0.40% |
| 4–5 | 23.512 | 21.20% | 9–10 | 14.961 | 13.50% |

## 7.3 Resources and vulnerability markets

In this section, the current state of vulnerability markets is presented. To this extend, it is important to briefly present the distribution of vulnerabilities based on CVSS and analyze their markets' types. According to the taxonomy proposed in [6, 127] there are primarily three types of stakeholders:

- *Vulnerability producers:* this includes freelance discoverers/sellers as well as captive discoverers (i.e. researchers, organization employers etc.).
- *Vulnerability markets:* this includes both regulated and unregulated markets.
- *Vulnerability consumers:* this refers to the taxonomy of attackers presented in Section 7.1.

The correlation among the above stakeholders is illustrated in Figure 7.1. It is shown that security companies (employees) can possibly have ties to unregulated markets and sell vulnerabilities having been found while performing their daily job operations (e.g. penetration testing) under the regulated framework (grey hat hackers). Furthermore, attackers of type *SK & CP* can take part in bug bounties in the context of *reward programs*, depending on their skills.

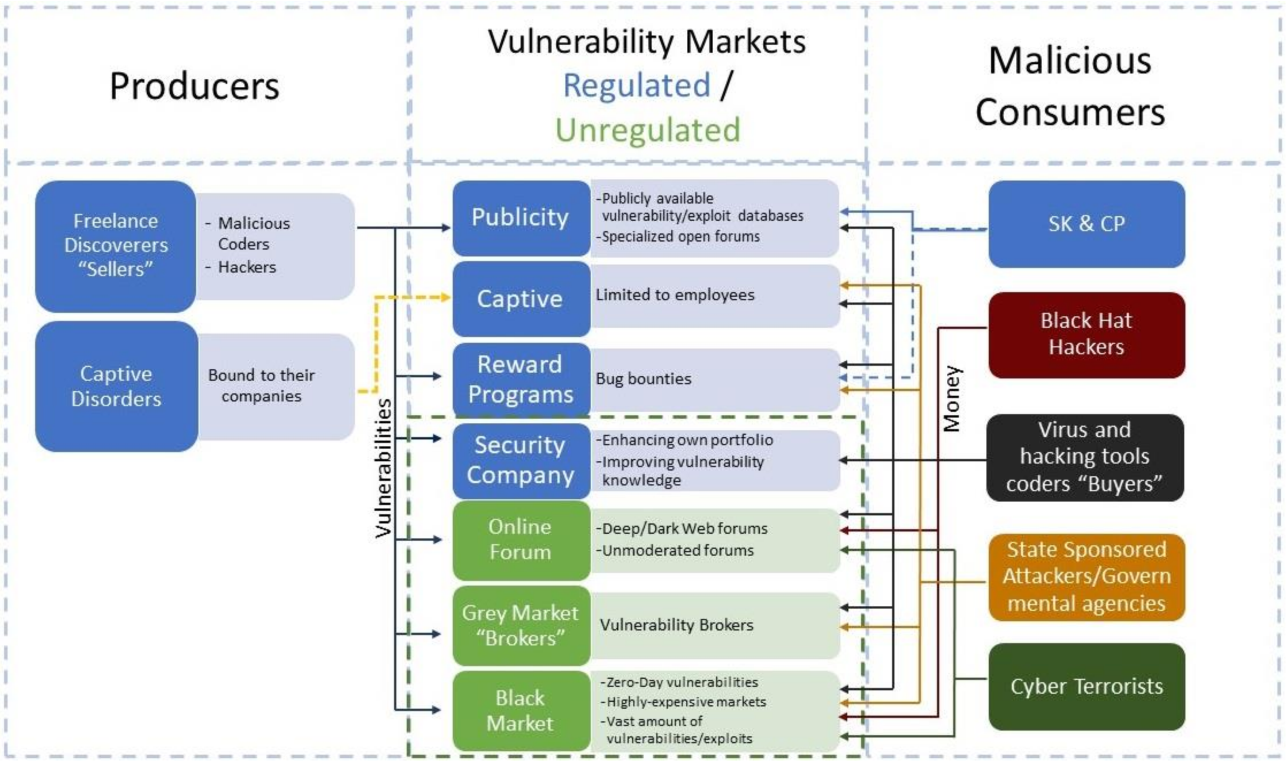


Figure 7.1: Vulnerability markets and attackers

The regulated and unregulated types of vulnerability/exploit markets are further described in the following sections.

### 7.3.1 Regulated markets' value

Regarding the regulated markets it is important to discuss the Reward programs in order to gain a clear view on the price range of vulnerabilities. These are bounty programs founded by companies, like Google, Apple, Microsoft, United Airlines and Master–card, governmental institutions, like the US Pentagon, and academic institutions, like MIT [81]. As an example, Google has paid approximately 12M USD during 2010–2018, while the largest single payout took place in 2017 and reached the 125K USD [35]. Furthermore, there are companies, like HackerOne, which provide bug bounty and vulnerability disclosure platforms and organize bug bounties for their clients; as of December 2017, they have paid in total more than 23.5M USD in bug bounties [39]. On the other hand, as shown in Figure 7.1, there are companies operating as vulnerability brokers that buy zero–day exploits, like Zerodium. From 2015 they are publishing a price list regarding zero–day exploits and they offer up to 1.5M USD per submission (*see* Figure 7.2).

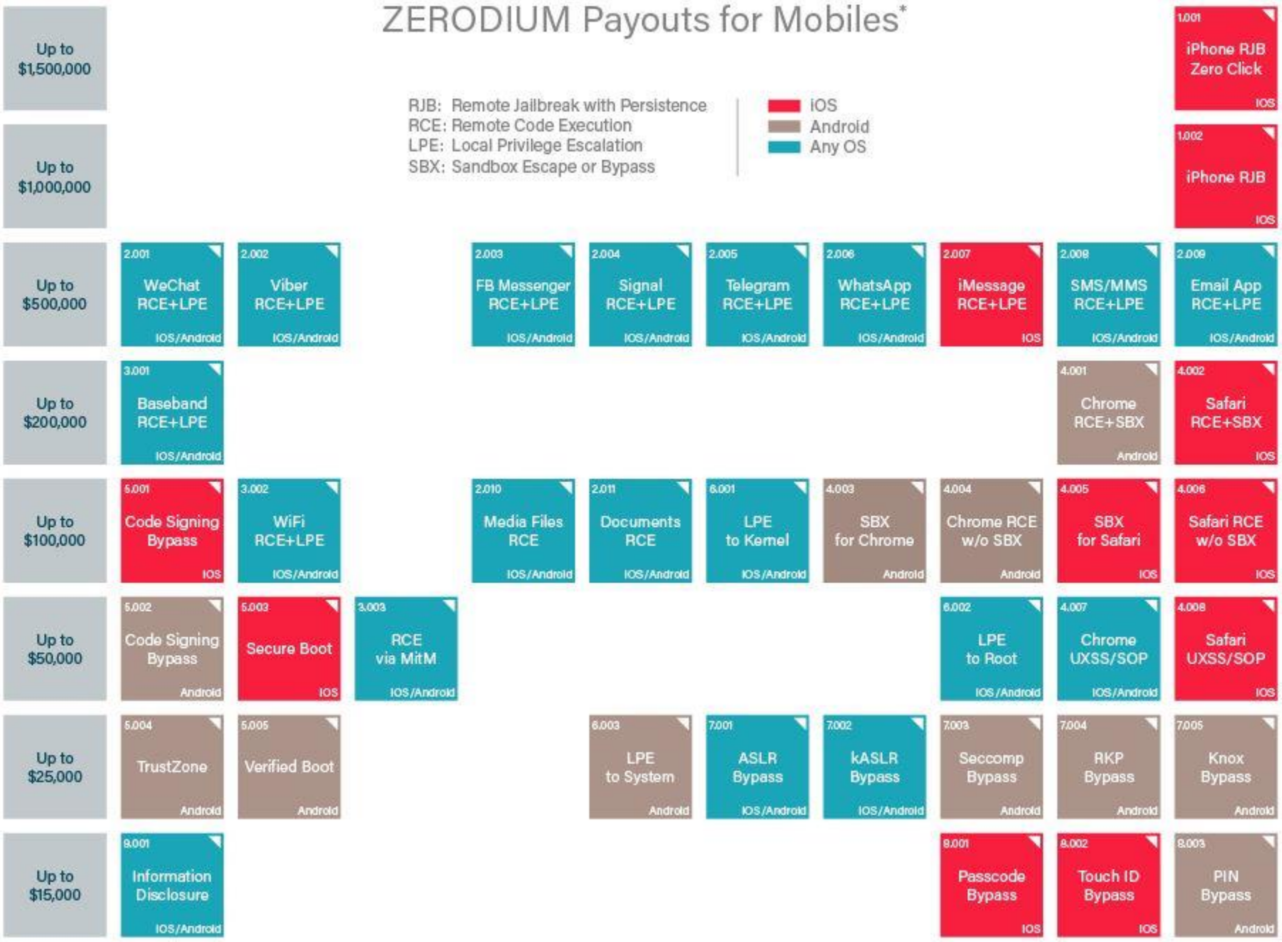

ZERODIUM Payouts for Mobiles*
RJB: Remote Jailbreak with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass
iOS
Android
Any OS
Up to $1,500,000
1.001 iPhone RJB Zero Click iOS
Up to $1,000,000
1.002 iPhone RJB iOS
Up to $500,000
2.001 WeChat RCE+LPE iOS/Android
2.002 Viber RCE+LPE iOS/Android
2.003 FB Messenger RCE+LPE iOS/Android
2.004 Signal RCE+LPE iOS/Android
2.005 Telegram RCE+LPE iOS/Android
2.006 WhatsApp RCE+LPE iOS/Android
2.007 iMessage RCE+LPE iOS
2.008 SMS/MMS RCE+LPE iOS/Android
2.009 Email App RCE+LPE iOS/Android
Up to $200,000
3.001 Baseband RCE+LPE iOS/Android
4.001 Chrome RCE+SBX Android
4.002 Safari RCE+SBX iOS
Up to $100,000
5.001 Code Signing Bypass iOS
3.002 WiFi RCE+LPE iOS/Android
2.010 Media Files RCE iOS/Android
2.011 Documents RCE iOS/Android
6.001 LPE to Kernel iOS/Android
4.003 SBX for Chrome Android
4.004 Chrome RCE w/o SBX Android
4.005 SBX for Safari iOS
4.006 Safari RCE w/o SBX iOS
Up to $50,000
5.002 Code Signing Bypass Android
5.003 Secure Boot iOS
3.003 RCE via MitM iOS/Android
6.002 LPE to Root iOS/Android
4.007 Chrome UXSS/SOP iOS/Android
4.008 Safari UXSS/SOP iOS
Up to $25,000
5.004 TrustZone Android
5.005 Verified Boot Android
6.003 LPE to System Android
7.001 ASLR Bypass iOS/Android
7.002 kASLR Bypass iOS/Android
7.003 Seccomp Bypass Android
7.004 RKP Bypass Android
7.005 Knox Bypass Android
Up to $15,000
8.001 Information Disclosure iOS/Android
8.001 Passcode Bypass iOS
8.002 Touch ID Bypass iOS
8.003 PIN Bypass Android

Figure 7.2: Zerodium mobile devices 0–day exploits price list

To summarize, the prices in the regulated markets range from few thousands up to 1.5M USD based on the characteristics of the vulnerability/exploit. As the numbers indicate, it is a profitable market. Nevertheless, one has to be very skillful to identify a vulnerability or an exploit that will be bought for high price.

### 7.3.2 Unregulated markets' value

The unregulated markets are actually divided in Gray markets and Black markets. It is highly difficult to find and access unregulated markets, especially in the Dark web as they tend to keep the vulnerabilities private. Governmental agencies are using this market (especially Gray markets) to buy and use vulnerabilities for both offensive and defensive purposes [34]. Thus, researching regarding the pricing of zero–day vulnerabilities and exploit kits it is not an easy task and only few information can be found (and not necessarily up to date). In most occasions are based on [9], [127], [34] a single zero–day vulnerability can be found from 20.000 USD to 100.000 USD while in few occasions can go up to 150–300K USD [1]. Table 7.4 provides an overview of the price list of exploit kits from 2011 up until 2013 [1].

Table 7.4: Price of exploit kits over time

| Exploit kit | Price (USD) | Year |
|---|---|---|
| Katrin | 25 daily | 2011 |

| | | |
|---|---|---|
| Robopak | 150 weekly or 500 monthly | 2011 |
| Blackhole v1.1.0 | 1.5K | 2011 |
| Blackhole v1.2.1 | 700 quarterly or 1.5K annually | 2011 |
| Bleeding Life v3.0 | 1K | 2011 |
| Phoenix v3.0 | 2.2K/2.7K per single/multithreaded domain | 2011 |
| Eleonore v1.6.3a and v1.6.4 | 2K | 2011 |
| Eleonore v1.6.2 | 2.5K–3K | 2012 |
| Phoenix (v2.3.12) | 2.2K per domain | 2012 |
| Styx sploit pack rental | 3K monthly | 2012 |
| Exploit kits that employ botnets | up to 10K | 2012 |
| CritXPack | 400 weekly | 2012 |
| Phoenix (v3.1.15) | 1K–1.5K | 2012 |
| NucSoft | 1.5K | 2012 |
| Blackhole–hosting (incl. crypter, payload, and source code) | 200 weekly or 500 monthly | 2013 |
| Whitehole | 200–1.8K rent | 2013 |
| Blackhole–license | license 700 quarterly or 1.5K annually | 2013 |
| Cool (incl. crypter and payload) | 10K monthly | 2013 |
| Gpack, Mmpack, Icepack, Eleonore | 1K–2K | 2013 |
| Sweet orange | 450 weekly or 1.8K monthly | 2013 |

Furthermore, Table 7.5 provides documented sales from 2013 to 2016. As it is depicted among the buyers are Governmental agencies and hacking teams [81]. It is important to highlight that the information disclosed in the table might refer to transactions that took place in both regulated and unregulated markets.

Table 7.5: Zero–day sales [81]

| Buyer | Seller | Price (USD) | Date |
|---|---|---|---|
| US LEA | Exodus intelligence | N/A | Nov. 2016 |
| FBI | Unknown | 1.3M | Apr. 2016 |
| Zerodium | Unknown | 1M | Nov. 2015 |
| Hacking team | Netragard | 105K | June 2015 |
| Hacking team | Eugene Ching (cyber researcher for Singaporean army) | 20K | Apr. 2015 |
| Hacking team | Netragard | 215K | Nov. 2014 |
| Hacking team | Netragard | 80.5K | July 2014 |
| Hacking team | Vitaliy Toropov | 40K | Feb. 2014 |
| Hacking team | Vitaliy Toropov | 45K | Oct. 2013 |

It is evident from the above information that critical zero–day vulnerabilities and exploits are very expensive to buy, as a unique set of technical skills is required for their identification. Thus, only elite attackers would be able to identify vulnerabilities and create exploits, while only attackers with enough money would be able to obtain critical vulnerabilities/exploits (e.g. state–sponsored attackers).

# 8. Cyber–Trust related scenarios

In the previous sections we have analyzed attackers' profiles, tools available for protecting networks from attacks as well as graphical security models to be utilized in an intelligent intrusion response system. In this section we focus on attacks applicable on Smart Homes and Mobile Devices as these are the main pillars of the project's pilots. Hence, the need to better understand common threats in these domains arises, along with the available tools for setting up a realistic simulation environment in order to test our research ideas and the performance of the prototype methods before being validated in the pilot.

## 8.1 Typical CT domain models

*Smart homes* and *mobile devices* are the domains where Cyber–Trust will be validated based on deliverable D2.3. To this end, this section explores the devices that networks of these domains typically include in order to get a better understanding on how a realistic simulation environment can be setup.

### 8.1.1 Smart home domain

The components that need to be chosen in this domain include end–user devices, their operating systems (OS), routers, services, versions, etc. [69]. In addition, there is a number of factors that should be considered for setting up the smart home network within IoT [71]; these are the network type (wired, wireless or both), the number of devices within the smart home, how these devices are connected to the IoT network [14].

Regarding the devices, the smart home's router is the first device that to be considered [8]; these routers, which are provided by ISPs with a built–in access point, must have high performance (number of packets per second) and be easy to manage [29]. Examples that could be used in the simulation environment include:

- *Unifi USG:* designed to be compatible with UniFi Enterprise Systems to provide routing and security to a home network[100]. It has three Gigabit Ethernet ports and the ability to route up to 1M packets per second. The device supports features, like Advanced Firewall, QoS, VLAN support and VPN.
- *Netgear N900:* has VPN support, which is compatible with Time Machine, and features USB storage access. Therefore, it is possible to access a connected USB hard drive from the network (or e.g. a smart TV)[101].
- *Google Wifi system:* aims to provide enhanced WiFi coverage by setting up multiple WiFi devices in a smart home. The router offers 802.11ac connectivity with 2.4GHz, 5GHz channels, 2x2 antennas, with support for beamforming. It also has two gigabit Ethernet ports, and contains a quad–core processor with 512 MB RAM and 4 GB flash memory[102].

Switches are typically part of the smart home network by connecting devices in a wired manner; they receive, process, and forward data to the destination [137]. The number of ports in a switch depends on the smart home's size and the number of devices to be connected (a port is devoted to the router [110]); common network devices that could be connected via switches are given below

- Access points (per floor)
- Network attached storage (NAS) drives / external hard drives
- Smart TV
- Game console
- Smart thermostat (some connect over a WiFi connection, whereas others need a bridge)
- Personal computer / other office peripherals

[100] https://www.ubnt.com/unifi/unifi–ap/
[101] https://www.netgear.com/
[102] https://store.google.com/product/google_wifi

A number of switches are being considered for Cyber–Trust’s simulation environment, like Unifi US–8–60W and Netgear ProSafe series. The above are next illustrated in the indicative setup of Figure 8.1.

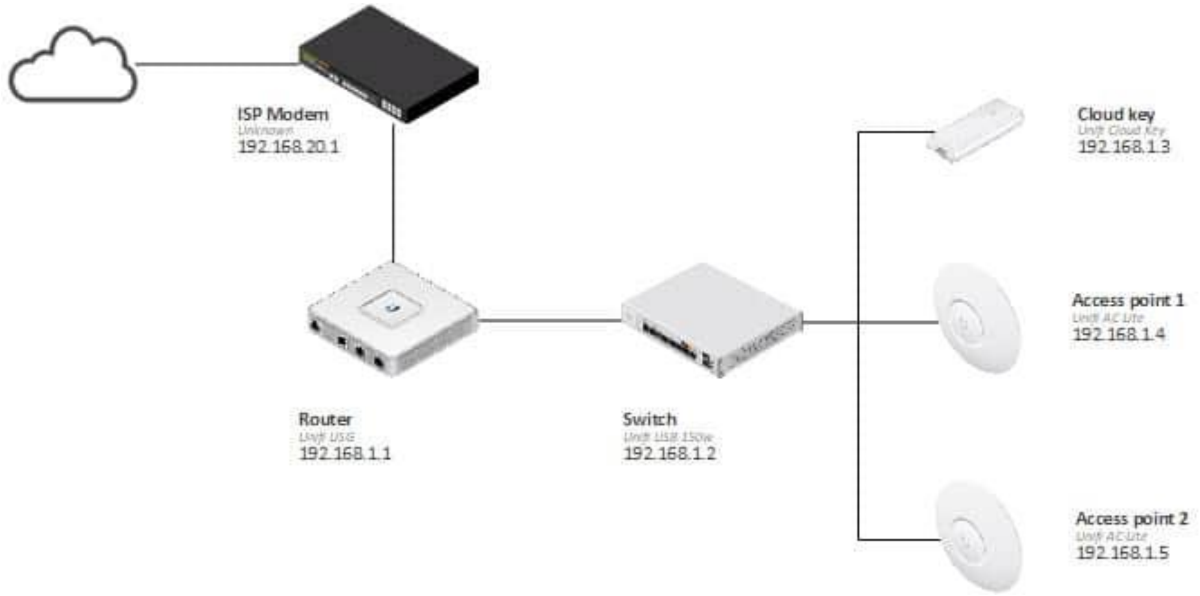


Figure 8.1. A smart home’s typical network setup

The combination between wired and wireless smart home networks in the context of Cyber–Trust provides a number of advantages, including the ability of modeling more complex environments using a multitude of protocols. Wired networks provide better connectivity than wireless networks, since the distance from the access point leads to performance degradation, and are less vulnerable to security issues. On the other hand, wireless networks offer a big advantage when it comes to mobility [73].

### 8.1.2 Mobile device domain

Cellular networks have developed to process and deal with a huge amount of data. They can also be used to connect physical things together like sensors, smartphones [15]. Communication across a cellular network is enabled by the transceivers and is packet–based; A mobile device could contain many transceivers, thereby having the capability to communicate over different radio networks (GPRS, Bluetooth, GSM, UMTS, LTE, Wi–Fi, etc.). For example, a cellular phone can include a GPRS transceiver for communicating with a cellular base station, a Wi–Fi transceiver for communicating with a Wi–Fi network, and a GPS transceiver for receiving a signal from a positioning satellite. A network typically includes a variety of elements that host logic for the tasks on the network. In modern packet–based networks, servers be scattered at various logical points on the network [117]. Servers might also be in communication with databases and can enable communication devices to access the contents of a database. A server can span several network elements, including other servers in the cellular network. The devices that can be connected to the cellular network are varied based on their company such as:

- *Apple iPhone:* it runs iOS and connectivity options include Wi–Fi, GPS, Bluetooth, NFC, Infrared, FM, 3G and 4G.
- *Samsung*, *LG*, *Sony*, and *HTC:* they run Android OS and connectivity options are as above. Moreover, the *HTC* device includes a number of sensors, such as a compass/magnetometer, proximity sensor, accelerometer, ambient light sensor and a gyroscope.

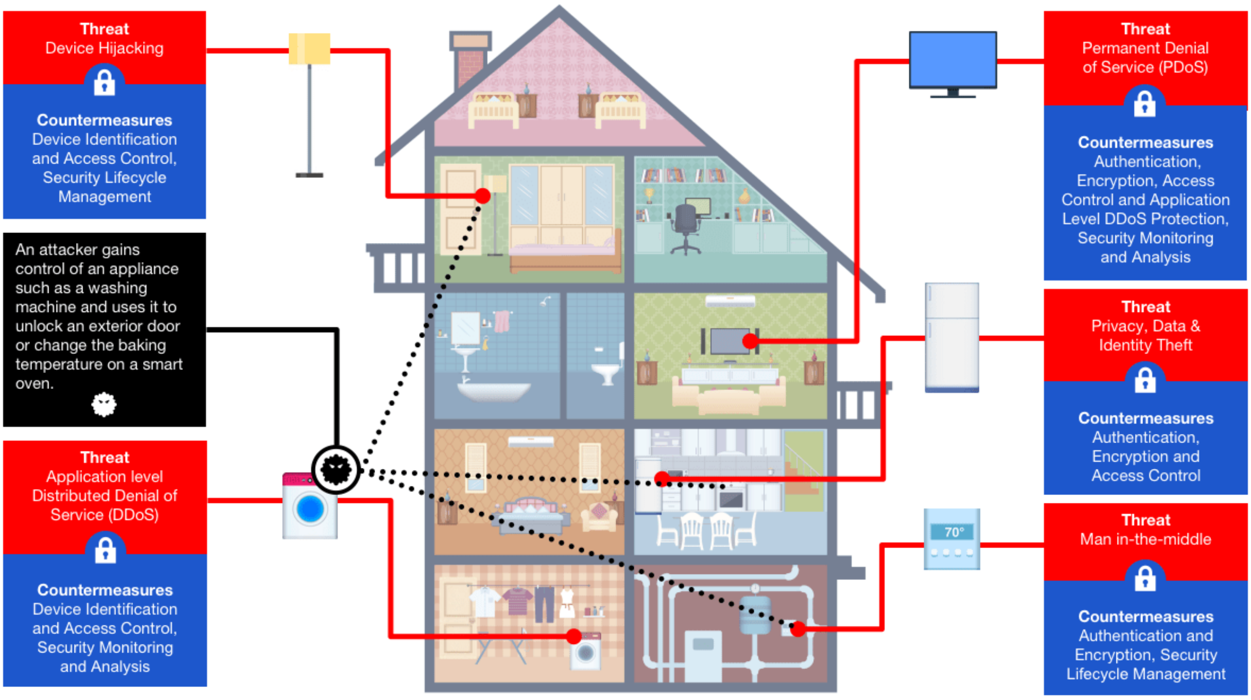
The above different devices can be connected in a *machine to machine* (M2M) fashion, which is typically the case in large networks of heterogonous devices that serve time–critical applications [77]; the goal is to keep these devices secure and safe from sophisticated attackers.

## 8.2 Typical attackers' strategies

Following Section 8.1, where we focused on the devices that a smart home and a mobile network can contain, we next explore the typical strategies that attackers might apply in these domains.

### 8.2.1 Smart home domain

Since the late 1970s, several studies have been devoted towards shaping the idea of a smart home [57]. This was facilitated by the advancements in the consumer electronics industry and the increase of the internet connectivity [88]. Living in a smart home environment provides a lot of advantages, ranging from economic profits to the improvement of smart home owners' daily lives.


Threat
Device Hijacking
Countermeasures
Device Identification and Access Control, Security Lifecycle Management
Threat
Application level Distributed Denial of Service (DDoS)
Countermeasures
Device Identification and Access Control, Security Monitoring and Analysis
Threat
Permanent Denial of Service (PDoS)
Countermeasures
Authentication, Encryption, Access Control and Application Level DDoS Protection, Security Monitoring and Analysis
Threat
Privacy, Data & Identity Theft
Countermeasures
Authentication, Encryption and Access Control
Threat
Man in-the-middle
Countermeasures
Authentication and Encryption, Security Lifecycle Management
70°

An attacker gains control of an appliance such as a washing machine and uses it to unlock an exterior door or change the baking temperature on a smart oven.

Figure 8.2. Typical Attacker's Strategies on Smart Home [116]

Security is a critical factor in this area [143]. Currently, numerous security issues have been reported, with about 80% of smart home devices being vulnerable to a wide range of attacks [28]. Obviously, connecting smart devices, e.g. smart door locks and fridges, led to several cyber security hazards; even connected child monitors are vulnerable to cyber-attacks [17]. The influence of each attack differs to a great extent due to a number of factors, like the ecosystem, the device and environment, and the available protection level, and attackers could disclose users' confidentiality or privacy [119]. Attackers' typical strategies on smart homes are depicted in Figure 8.2.

### 8.2.2 Common cyber–security threats and attacks against smart home devices

#### *8.2.2.1 Botnets*

A botnet is a network of systems aiming at remotely taking control and distributing malware [38]. The botnet operator takes the control through *command and control servers* (C&C Server). Criminals may use them for stealing private information, exploiting online–banking data, performing DDos–attacks or sending spam and phishing emails [115]. With the recent growth of the IoT, many objects and devices are in threat or part of the so–called thingbots (a botnet that combines independent connected objects). Botnets along with thingbots consist of various different connected devices, including computers, laptops, smartphones, tablets, and other smart devices, and are hard to identify [79]. These things have two key features in common, they are internet enabled and they are able to transfer data automatically through a network [82]. Generally, the aim of a botnet is to flood a target system with a vast number of requests in order to exceed its capacity in serving these requests, thus resulting into a denial of service to legitimate users.

#### *8.2.2.2 Man–In–the–middle*

The idea of man–in–the–middle attacks is that the attacker intercepts and breaches the communications between two systems [144], which are confident that are communicating directly with each other. As the attacker controls the main communication, the receiver is misleaded into thinking that received messages are legitimate [38]. Within this area, many cases have now been conveyed by smart home owners, including cases of hacked vehicles and hacked smart refrigerators [36]. Because of the nature of the devices being hacked, these attacks can be quite harmful on a smart home's devices. These devices can be anything from industrial tools, machinery, or vehicles to harmless connected ones like smart TV's or garage door openers [67]. Generating fake temperature data, using an environmental monitoring device, and sending these data to the cloud is an example of attack. Likewise, a hacker may deactivate vulnerable HVAC systems throughout a heat wave, producing a disastrous situation for service providers with affected models.

#### *8.2.2.3 Data and identity theft*

This kind of data is created using insecure devices such as wearables and smart appliances providing cyber attackers with a huge amount of targeted information that can be subjugated for fraudulent transactions and identify theft [136]. Even though the news is full of frightening and unpredictable hackers accessing data and money with all kinds of remarkable hacks, the users themselves are mostly the main enemy to their security [130]. Devices connected over the internet, such as iPad, Kindle, smartwatch and locks, whose protection has been neglected present very easy targets to thieves and opportunistic finders [132]. They key to achieve a theft is to collect many data with patience so that they can be used against the owner of the hacked device [82]. The hacker usually combines many resources in order get outstanding idea of the personal identity of the smart device user, including the general data available over the internet, social media information, data from smart watches, fitness trackers, smart meters, smart fridges and many possible means [18]. In general, the more information can be discovered about a device owner, the easier and the more advanced a targeted hack aimed at identity theft can be [2].

#### *8.2.2.4 Social engineering*

Social engineering refers to the way an attacker uses to manipulate the users so as to provide confidential and private information [37]. The criminals are looking for many types of information of the targeted victim, but the attacker typically deceives the users into sending passwords or bank information [40]. Alternatively, the criminals might try to access a computer and install software that will provide them access to personal and private information, on top of giving them full control over the user's computer [28]. The key method that usually used in the social engineering hacks is the phishing emails. Through these emails, the hacker tries to guide the users to divulge their information, or redirect them to websites like banking or shopping sites that look legitimate, enticing the users to enter their details [143].

#### 8.2.2.5 *Denial of service attack*

A *denial–of–service* (DoS) attack tries to make a machine or network resource temporarily inaccessible to its intended users or persistently damaging services of a host connected to the Internet [21]. There are many reasons for unavailability; however, it typically refers to infrastructure that cannot cope because of capacity overload [67]. In a *distributed denial of service* (DDoS) attack, a vast number of malicious systems gather to attack one target [115]. Usually this attack is achieved using a botnet, where several devices are set up to simultaneously request for a service [79]. Therefore, in the DDoS case, incoming traffic that floods a target originates from multiple sources, thus making it very hard to stop the cyber–attack by merely blocking a single source. Actually, because of the lack of security in smart home devices, the studies showed that the percentage of the DDoS attacks have doubled from 3% to 6% in 2016 [79]. This duplication is not astonishing, particularly in the case of one compromised smart sensor on a network is able to infect many similar devices running the same software. Therefore, these infected devices are forced to join huge botnet armies that implement crippling DDoS attacks [82].

#### 8.2.2.6 *Device hijacking*

The attacker hijacks and effectively undertakes the control of a device [57]. This kind of attacks are difficult to discover, since the basic functionality of the device is not changed by the attacker. Furthermore, it is very likely to infect all the smart devices in the home through merely taking one device of them. For instance, a hacker who initially compromises a thermostat is able to theoretically get control over the entire network of the smart home and consequently can remotely unlock a door or change the keypad PIN code to limit entry.

### 8.2.3 Mobile device domain

Security threats in mobile devices are growing. In 2017, Kaspersky Lab reported that they detected more than 5M malicious installation packages, and more than 500K mobile ransomware Trojans [147]. This shows that attackers have an increasing interest in using today’s mobile devices for spreading mobile malware that steals user’s information, bombards our devices with unwanted ads, and can even be used to launch other types of attacks such as denial of service attacks. The rest of the section highlights common mobile threats and attackers’ strategies on modern mobile devices and other handheld devices.

### 8.2.4 Common cyber security threats and attacks against mobile devices

#### 8.2.4.1 *Zero–day vulnerabilities*

A *zero–day vulnerability* is defined as a software vulnerability whose existence was unknown (thus, no patch or fix has been released) and it is discovered during the process of a security incident’s post–analysis [34]. A zero–day vulnerability is one of the most challenging attack vectors to detect, as the attacker might develop an exploit to compromise a mobile device that still vulnerable and unpatched. Attackers develop software exploits to take advantage of security vulnerabilities. Such malicious software can compromise a vulnerable mobile device and enable the attacker to control the device entirely. In most cases, a patch from the software developer can fix this. However, when a mobile device becomes infected, exploit malware can steal its data, allowing hackers to take unauthorised control of the device [142]. Figure 8.3 shows the typical life of a zero–day vulnerability starting from an attacker discovering the vulnerability and developing an exploit to take advantage of this unpatched/unknown venerability.

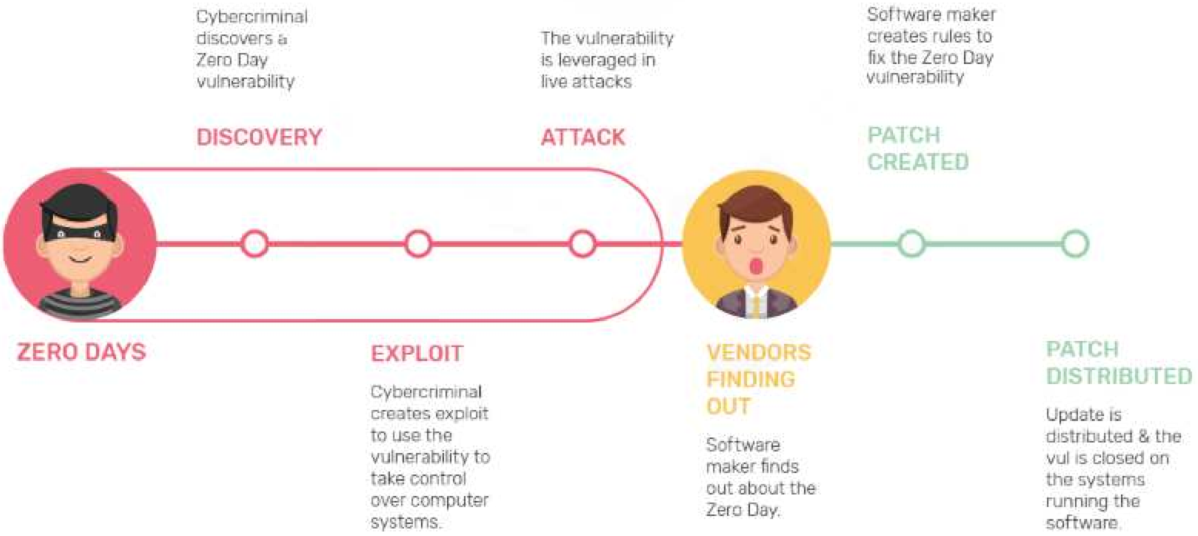

Cybercriminal discovers a Zero Day vulnerability
DISCOVERY
The vulnerability is leveraged in live attacks
ATTACK
Software maker creates rules to fix the Zero Day vulnerability
PATCH CREATED
ZERO DAYS
EXPLOIT
Cybercriminal creates exploit to use the vulnerability to take control over computer systems.
VENDORS FINDING OUT
Software maker finds out about the Zero Day.
PATCH DISTRIBUTED
Update is distributed & the vul is closed on the systems running the software.

Figure 8.3. Life–span of a typical zero–day vulnerability [141]

This is followed by the application of the actual attack on the victim's vulnerable device unless the provider of the vulnerable device/software releases a patch fix.

#### *8.2.4.2 Malware and spyware*

A malware is a malicious piece of software developed specifically for stealing information, harm an electronic device or to propagate itself and control devices it infected.

Mobile devices can be infected by mobile malware and spyware in various attack vectors; these include installing legitimate applications that were modified with malicious payloads, getting drive–by downloads, etc. The infected software will perform at least one of the following techniques, namely privilege escalation, remote control, financial control, and intelligence gathering, which provide an attacker with a variety of options to utilise a compromised mobile device [111]. It is possible to know if a desktop computer is infected with malware, as there are symptoms that sometimes are noticeable such as slowing down the performance of the computer, starting popping–up fake ads, and sometimes the computer crashes unexpectedly, the fan starts whirring noisily and unfamiliar icons show up in the desktop [113]. However, it is more challenging to know whether a mobile device such as Android or iOS phones is infected with malware. According to Porta, the following are the common forms of recent mobile malware [113]:

- *Adware* – shows frequent ads to a user in the form of pop–ups, sometimes leading to the unintended redirection of users to web pages or applications
- *Banker malware* – attempts to steal users' bank credentials without their knowledge
- *Ransomware* – demands money from users and, in exchange, promises to release either the files or the functionality of the devices being 'held hostage'
- *Rooting malware* – 'roots' the device, essentially unlocking the operating system and obtaining escalated privileges
- *SMS malware* – manipulates devices to send and intercept text messages resulting in SMS charges. The user is usually not aware of the activity
- *Spyware* – monitors and records information about users' actions on their devices without their knowledge or permission
- *Trojan* – hides itself within a piece of seemingly innocent, legitimate software.

#### 8.2.4.3 Botnets

One of the most devastating type of malware that infects mobile devices is botnets. A botnet infecting machines worldwide, receives commands from their bot–master that has full control over the infected device and launches illegal actions such as DDoS, credential stealing, spam sending, bank account and credit card theft and downloading other malware [139].

#### 8.2.4.4 Keylogger

Keyloggers and screenloggers are applications that can capture, store and send active device screens, without the attention of device's owner. Currently available keyloggers are considered genuine applications and they are used to do many legitimate and legal functions, such as tracking children's use of the internet; however, many cases of inappropriate use in business environments have been reported [131]. Typical keyloggers search for specific events or unique keys to identify sensitive and confidential information that is next sent to the adversary. For instance, when a mobile user enters a username or an email address, the spyware can recognize such activity as filling a login form, in which keying in a password will follow. Looking for a special event or a particular key is much easier than trying to infer each entered key. This approach can be used in taps inference by looking (i.e., in the stream of sensor data) for a specific symbol(s) (e.g., @ key, possibly followed by Next button to indicate a subsequent e–mail password) or for an interesting event, such as a system start–up, launching of a password–protected session/app, or even the start of a phone conversation where valuable information, such as PIN, social security number, and date of birth may be requested [46].

#### 8.2.4.5 Wireless attack

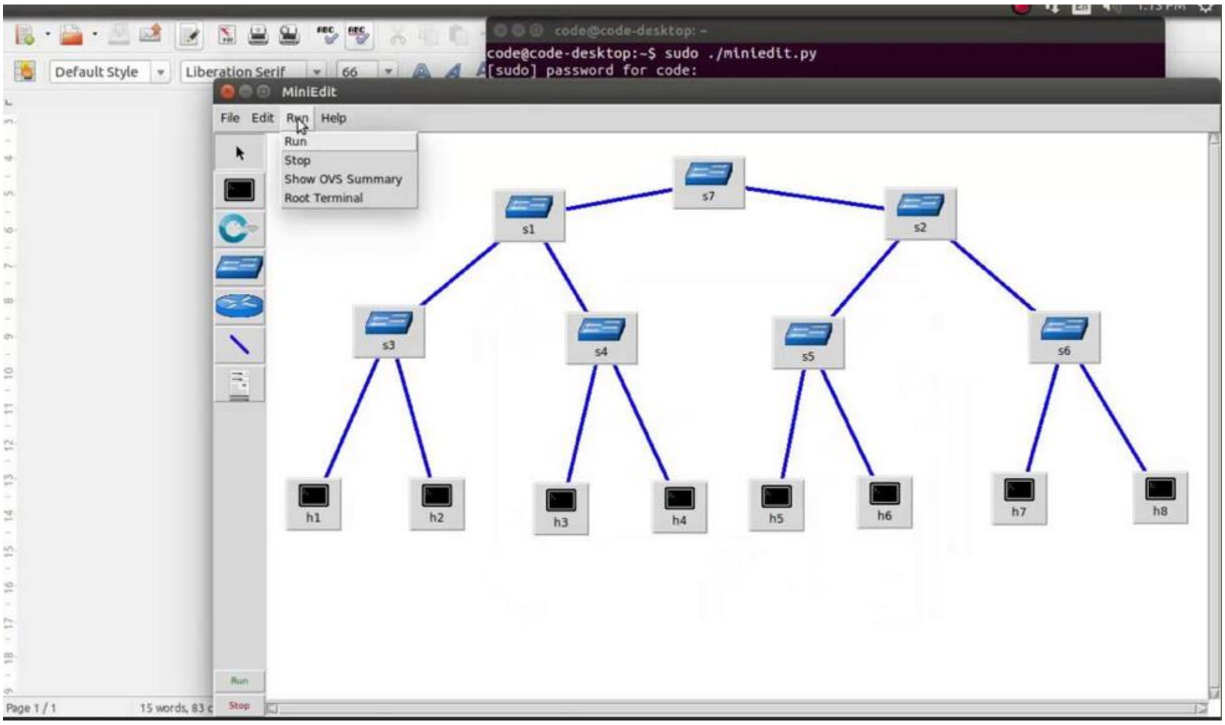
Hackers can attack wireless network users to intercept transmitted Wi–Fi traffic between mobile devices and wireless access points, and even alter the intercepted traffic to inject malware into websites being read by the mobile device user. Security analysts discovered many security vulnerabilities on mobile devices that take advantage of wireless implementations, where Android and Linux–based devices are affected the most by multiple vulnerabilities [70, 125, 135]. Further, standard Wi–Fi networks security measures such as using WPA or WPA encryption, have known weaknesses that affect the operating system, incl. macOS, Windows, iOS, Android, and Linux devices rendering them vulnerable. Intercepting traffic allows attackers to read information that was previously assumed to be safely encrypted, and hackers do not need even to crack a Wi–Fi password to achieve this. The vulnerability requires that a device be in range to a malicious attacker, and it can be used to steal credit card numbers, passwords, chat messages, photos, emails, and lots of other online communications [151].

## 8.3 Simulation environment

The provision of a simulation platform is via Docker coupled to VMWare virtualization in order to yield a scalable, controllable IoT simulator capable of meeting the requirements of Cyber–Trust. A simulator is designed to have a resemblance to the actual network, but only simulate functions within the network, such as normal device operating behavior and traffic versus attack scenarios (DDoS, malware executables, etc.).

Towards deciding the most technically sound approach to delivering a simulation capability, three market–leading, manufacturer agnostic capabilities were examined: two of which only function as network simulators whereas a third one serves as a containerization capability that can be exploited as a network simulator. The capabilities assessed are GNS3, Mininet and Docker. The first two are GUI–driven and Docker is focusing on CLI functionality (with some third–party OS–specific GUI capabilities available). The overall analysis is summarized in Table 8.1, and discussed in–depth in the sequel.

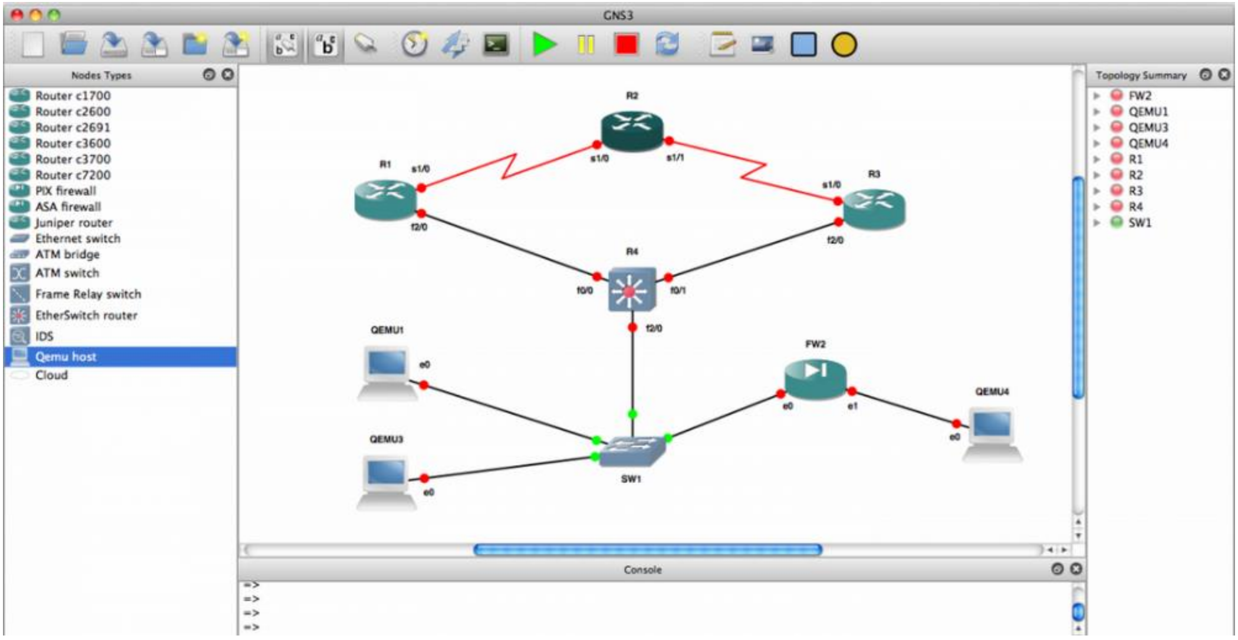
Mininet is designed to research and teach networking, including *software–defined networks* (SDN). It creates a flat ethernet network of multiple OpenFlow–enabled Ethernet switches and multiple hosts connected to these switches. Custom topologies are driven by user–generated Python scripts that provide the user with a great deal of flexibility in terms of network topology, but most importantly with the ability to transition to a real–world system.



```
code@code-desktop:~$ sudo ./miniedit.py
[sudo] password for code:
```

Figure 8.4. Mininet GUI

GNS3 is a GUI–driven network simulator that allows the user to run multiple emulated systems centered on the Cisco *Internetwork Operating System* (IOS), which is a commercial–license–driven Cisco provision. It is very powerful, allowing the emulation of Cisco IOSs on Windows or Linux based computers. Emulation is possible for a long list of router platforms and PIX firewalls. Using an EtherSwitch card in a router, switching platforms may also be emulated to the degree of the card's supported functionality. The reliance though on commercial licensing to deliver core functionality makes it rather unsuitable for the project's needs.



```
=>
=>
=>
=>
```

Figure 8.5. GNS3 GUI

Docker diverges from the pure simulation capabilities discussed above, as it provides a system capability rather than a simple piece of software. Docker is an extension of the Linux containerization protocols within the Linux kernel. Individual capabilities such as software, hardware OS and general operating systems are 'containerized' to allow the user to build containerized applications that deliver portability, service discovery, load balancing, security, performance and scalability. The core architecture is shown in Figure 8.6.

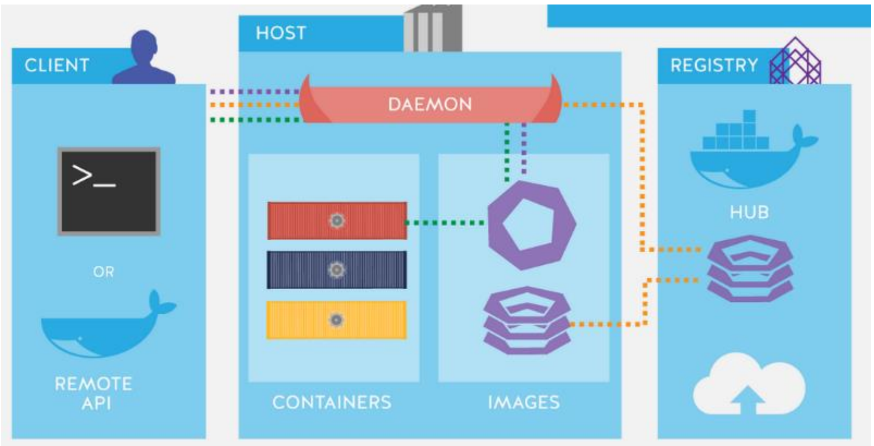


Figure 8.6. Docker architecture

Within these paradigms, the container networking model, which is shown in Figure 8.7, delivers the docker networking architecture interfaces that enable these paradigms to be delivered. The network model constructs are what allow Docker to be considered in the case of this deliverable as a suitable replacement for dedicated network simulation software, because whilst most computer–based components can be containerized and hosted on docker, the network model is the core element that allows these containers to function as a network.

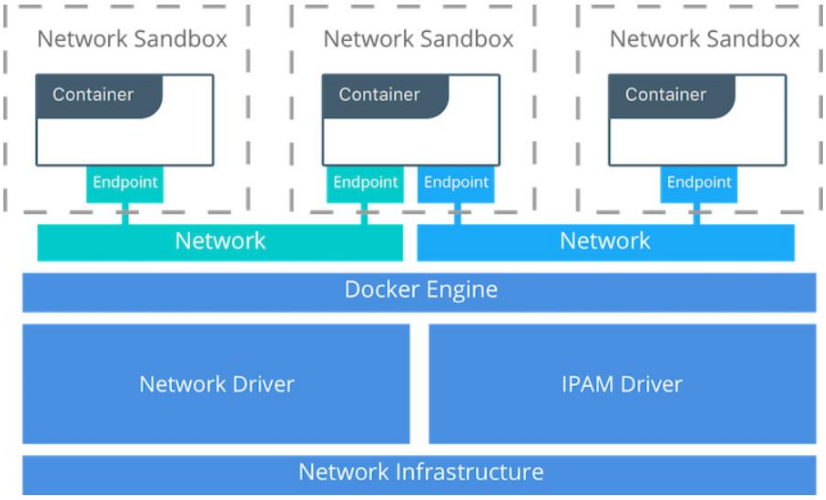


Figure 8.7. Container networking model

The network sandbox contains the configuration of a container's network stack. This includes management of the container's interfaces, routing table, and DNS settings. A sandbox may contain many endpoints from multiple networks and an endpoint joins a sandbox to a network. The network model does not specify a network in terms of the OSI model. An implementation of a network could be a linux bridge, a VLAN, etc. A network is a collection of endpoints that have connectivity between them. Two interfaces are provided: network drivers (native and remote) and IPAM drivers.

A vast array of containerized hardware and software exists on the open–source Docker Hub, and if specific capabilities are required the user can always containerize a specific capability themselves. Thus, Docker provides a scalable, adaptable, open–source networking capability free from the usability restrictions faced by users of specific software such as Mininet or GNS3.

In summary, as shown in Table 8.1, the three capabilities assessed all had their strengths, and Docker seems to be the best solution for the development of Cyber–Trust's simulator.

Table 8.1. High–level comparison of simulation environments

| Capability | Mininet | GNS3 | Docker |
|---|---|---|---|
| **Open Source** | Yes | Yes | Yes |
| **Windows Support** | Yes | Yes | Yes |
| **UNIX/Linux Support** | Yes | Yes | Yes |
| **Simulation mode** | No | Yes | No |
| **Emulation mode** | Yes | Yes | Yes |
| **Compatible with real–world controllers** | Yes | No | Yes |
| **Scalable** | Yes (but complex) | No | Yes |
| **Traffic Flow** | Yes | No | Yes |
| **Malware injection** | DDoS only | No | Yes |
| **Hardware agnostic** | Yes | Partly | Yes |

The requirement for a simulation platform is driven by the user needs to simulate the scenarios defined in deliverable D2.3. At this stage, we focus on the provision of a simulation capability that covers the first use case domain of D2.3.

**Smart Home Domain (SHD).** The definition of the Smart Home for the purpose of the simulation is a gateway, with an associated Intrusion Protection System, behind which a set of heterogeneous devices exist that cover the current market in terms of mobile devices and connected 'white goods' (thermostats, DVR, webcam, etc.). This simulation can be effectively summarized as covering the TCP/IP–focused network capabilities associated with IoT, including network protocols such as UDP, TCP and HTTP and underpinned by both legacy IPv4 and the IoT–enabling IPv6 protocols.

The SHD is, for the purposes of simulation, not a home per–se, but rather a connected set of capabilities against which the following actions can be conducted:

- Normal traffic injection (scapy);
- DDoS injection: High–Orbit Ion Cannon (http) and Low Orbit Ion Cannon (tcp/udp);
- Malware injection (DB of malware executables).

This is achieved by means of virtualization, where the increased resource consumption demanded by virtualized environments is offset by the protection offered by *virtual machines* (VM) when conducting research into malware through isolation from the host hardware. It also allows a level of flexibility as to the virtualized capabilities included in the simulation (i.e. different IDPSs can be run to compare performance in differing scenarios). A traffic generator VM (Scapy), DDoS VM (LOIC & HOIC) and a malware DB VM will connect to an IDPS VM (Snort/Suricata) which will, in turn, connect to the Smart Home VM. The Smart Home VM will run a networked set of containers, with a container running the device under review, i.e. a smart meter.

# 9. Conclusions

This deliverable reviewed the various methodologies and tools that can be used to efficiently model possible attack strategies. A systematic approach to achieve this goal is related with modelling these strategies with the so–called graphical security models (e.g. attack trees/attack graphs), which allow for convenient representation of the possible steps that an attacker may follow towards his final goal, in conjunction with the privileges obtained at each step (or with the actual impacts that occur with respect to security). These models are based on appropriate information that needs to be acquired at the first place, such as information on network topology, on nodes/devices connectivity, as well as on vulnerabilities that exist; by these means, attack strategies are being systematically analyzed so as to be able to take proper decisions with regard to the mitigation measures that need to be implemented. Moreover, such an analysis is strongly related to a risk management on the overall system, by appropriately utilizing the probabilities of occurrence of the identified vulnerabilities in conjunction with their impact upon successful exploitation.

This report presented a detailed comparative study, in terms of well–defined criteria, of all the relevant tools and methodologies, whilst typical realistic scenarios within the framework of Cyber–Trust project are also given. The main outcomes of this report can be summarized as follows:

- Utilizing *attack graphs* seems to be the most suitable modelling strategy (although adopting a hybrid model consisting of both tree–based and graph–based models could be convenient in some cases).
- *Probabilistic attack graphs* (e.g. Bayesian attack graphs) provide also the means for performing risk analysis via systematically considering the attack probabilities (based on the relevant CVSS scores). Therefore, they will be considered in the framework of the Cyber–Trust project.
- The attack graph to be used needs to have certain properties so as to *interact with the intelligent intrusion response system (iIRS)*. Moreover, to cope with scalability issues, it is highly probable that *hypergraphs* need to be employed – e.g. associating each node of a graph with a cluster.
- *Nmap* and the *Angry IP Scanner* (both being open source) are the tools that will be used for acquiring information on the list of devices lying within Cyber–Trust's protection domain, as well as on network topology, ports detection, host reachability, security measures deployed (packets filtering, firewalls etc.) and versions detection. Such info will in turn feed the attack graph model.
- Whenever needed, the capabilities of the above tools may be complemented by other tools, such as *NetworkMiner*.
- The *Nmap* will also be used in the context of Cyber–Trust for detecting vulnerabilities and backdoors. Moreover, to this goal, the freely available *OpenVAS* tool will be also used, which also integrates well with the Nmap. Again, the information obtained from these tools will in turn feed the attack graph model.
- In case that a reconnaissance tool is needed in the context of processing information for feeding the attack model, then the open source ReconDog seems to be a right option, whilst the Spiderfoot – up to the extent that its license limitations allow – will be also considered.
- Risk assessment in the context of Cyber–Trust will be built upon dynamic approaches that allow to exploit measurable information available from security standards so as to automatically update risk models; such models also rely on attack graphs.
- Snort or Suricata will be used, possibly in combination with other tools, like Bastille, that complement their functionalities in order to enforce the mitigation actions at the host or network level.
- Attackers have different skill levels as well as different amount of budget to spent. Each attacker has been correlated with the CVSS metrics as well as with the different zero–day markets.
- The simulation environment will be a mixture of dockers and virtual machines on a VMWare Vsphere infrastructure, simulating smart home devices. The network connectivity of these devices will be provided through Mininet or GNS3 (if Cisco routers and switches need to be simulated).

# References

[1] Ablon, L., Libicki, M.C. and Golay, A.A., 2014. *Markets for cybercrime tools and stolen data: Hackers' bazaar*. Rand Corporation.

[2] M. A. N. Abrishamchi, A. H. Abdullah, A. D. Cheok, K. S. Bielawski, K. "Side channel attacks on smart home systems: A short overview", in *Proceedings IECON 2017 – 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017–January, pp. 8144–8149, 2017.

[3] F.–X. Aguessy, O. Bettan, G. Blanc, V. Conan, and H. Deba, "Bayesian Attack Model for Dynamic Risk Assessment," arXiv:1606.09042 [cs.CR] 2016.

[4] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, E. I. Tatli, "Automated Generation of Attack Graphs Using NVD", *8th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pp. 135–142, 2018.

[5] M. Albanese, S. Jajodia, A. Pugliese, V. Subrahmanian, "Scalable analysis of attack scenarios," in: V. Atluri, C. Diaz (Eds.), *European Symposium on Research in Computer Security – ESORICS 2011*, pp. 416–433, 2011.

[6] A.M. Algarni and Y.K. Malaiya. Software Vulnerability Markets: Discoverers and Buyers. International Journal of Computer, Information Science and Engineering Vol: 8, No: 3, 2014.

[7] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, graph–based network vulnerability analysis," in *Proc. of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, ACM, 2002, pp. 217–224, 2002.

[8] N. Apthorpe, D. Reisman, N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic", *arXiv preprint arXiv:1705.06805*, 2017.

[9] J. Armin and M. Cremonini. "0–Day Vulnerabilities and Cybercrime," In Proc. *10th International Conference on Availability, Reliability and Security*, pp. 711 – 718, 2015.

[10] M. Artz, "NetSPA : A network security planning architecture", Massachusetts Institute of Technology, 2002.

[11] D. Baca, K. Petersen, "Prioritizing countermeasures through the countermeasure method for software security (CM–Sec)," in: M.A. Babar, M. Vierimaa, M. Oivo (Eds.), *Product–Focused Software Process Improvement*, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 176–190, 2010.

[12] G. Barlett, J. Heidemann, C. Papadopoulos, "Understanding Passive and Active Service Discovery", in *Proc. of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07)*, New York, NY, USA, pp. 57–70, ACM, 2007.

[13] M. Barrèrre, E. C. Lupu, "Naggen: A Network Attack Graph GENeration Tool", *IEEE CNS 17*, pp. 378–379, 2017.

[14] D. Basu, G. Moretti, G. S. Gupta, S. Marsland, "Wireless sensor network based smart home: Sensor selection, deployment and monitoring", In *Sensors Applications Symposium (SAS),* IEEE, pp. 49–54, 2013.

[15] A. Birenboim, N. Shoval, "Mobility research in the age of the smartphone", *Annals of the American Association of Geographers*, vol. 106, no. 2, pp. 283–291, 2016.

[16] S. Bistarelli, F. Fioravanti, P. Peretti, "Defense trees for economic evaluation of security investments," in *Proc. of the First International Conference on Availability, Reliability and Security (ARES 2006)*, 2006, pp. 337–350.

[17] A. Brauchli, D. Li, "A solution based analysis of attack vectors on smart home systems", *2015*

*International Conference on Cyber Security of Smart Cities, Industrial Control System and Communications, SSIC 2015 – Proceedings*, pp. 1–6, 2015.

[18] M. L. R Chandra, B. V. Kumar, B. Sureshbabu, "IoT enabled home with smart security", *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, pp. 1193–1197, 2018.

[19] K. Coffey, R. Smith, L. Maglaras, H. Janicke, "Vulnerability Analysis of Network Scanning on SCADA Systems", *Security and Communication Networks*, Hindawi, 2018.

[20] Computer Security Institute, "2010/2011 Computer Crime and Security Survey", pp. 1–40, 2011. Available at: https://cours.etsmtl.ca/gti619/documents/divers/CSIsurvey2010.pdf

[21] L. Coppolino, V. Dalessandro, S. Dantonio, L. Levy, L. Romano, "My smart home is under attack", in Proceedings of 18th IEEE International Conference on Computational Science and Engineering (CSE 2015), pp. 145–151, 2015.

[22] Council of Europe, "Cyberterrorism: The Use of the Internet for Terrorist Purposes", United Nations Office on Drugs and Crime, 12(1), p. 497, 2007.

[23] CVE Details, "Current CVSS Score Distribution For All Vulnerabilities", Available at: https://www.cvedetails.com/cvss–score–distribution.php , [Accessed: 17 December 2018]

[24] Cyber–Trust D2.2, "Threat and Risk Assessment Methodology," page 40, 2018.

[25] M. Dhawan, R. Poddar, K. Mahajan, V. Mann, "Sphinx: Detecting security attacks in software–defined networks," in: *Proc. of the Network and Distributed System Security Symposium (NDSS 2015)*, pp. 1–15, 2015.

[26] K. Edge, "A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems Via Attack and Protection Trees," Ph.D. Thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, USA, AAI3305523, 2007.

[27] ENISA, "Good Practice Guide for Incident Management," ENISA, pp. 1–110, 2010.

[28] E. Fernandes, A. Rahmati, J. Jung, A. Prakash, "Security Implications of Permission Models in Smart–Home Application Frameworks", *IEEE Security and Privacy*, vol. *15,* no. 2, pp. 24–30, 2017.

[29] D. Geneiatakis, I. Kounelis, R. Neisse, I. Nai–Fovino, G. Steri, G. Baldini, "Security and privacy issues for an IoT based smart home" In *40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2017),* pp. 1292–1297, 2017.

[30] N. Ghosh, I. Chokshi, M. Sarkar, S. K. Ghosh, A. K. Kaushik, S. K. Das, "NetSecuritas: An Integrated Attack Graph–based Security Assessment Tool for Enterprise Networks", in *Proc. of the 2015 International Conference on Distributed Computing and Networking (ICDCN '15)*, New York, NY, USA, p. 30, ACM, 2015.

[31] G. Gonzalez–Granadillo, et *al.* "RORI–based countermeasure selection using the OrBAC formalism," International Journal of Information Security, vol. 13, no. 1, pp 63–79, Feb. 2014.

[32] G. Gonzalez–Granadillo, et *al.* "Selecting optimal countermeasures for attacks against critical systems using the attack volume model and the RORI index," *Computers & Electrical Engineering*, vol. 47, pp. 13–34, Oct. 2015.

[33] G. Gonzalez–Granadillo, E. Doynikova, I. Kotenko, and J. Garcia–Alfaro, "Attack Graph–Based Countermeasure Selection Using a Stateful Return on Investment Metric," *10th Int'l Symposium on Foundations and Practice of Security – FPS 2017*, LNCS 10723, pp. 293–302, 2018.

[34] D. Gritzalis, "Zero–Day Vulnerabilities: A Primer", 2017. Available at: https://infosec.aueb.gr/Publications/ICT Security 2017 Zero–Day website.pdf (Accessed: 12 December 2018).

[35] E. Griffith, 2017. "7 Huge Bug Bounty Payouts." PCMag, June 9. https://www.pcmag.com/feature/

354224/7–huge–bug–bounty–payouts.

[36] P. Gupta, J. Chhabra, "IoT based Smart Home design using power and security management", *2016 1st International Conference on Innovation and Challenges in Cyber Security (ICICCS)*, pp. 6–10, 2016.

[37] V. Ha, A. Peculea, "Secure and Extensible Smart Home Template", in *17th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–6, 2018.

[38] O. Hachinyan, A. Khorina, S. Zapechnikov, "A Game–Theoretic Technique for Securing IoT Devices against Mirai Botnet", *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 1500–1503, 2018.

[39] HackerOne, "The 2018 Hacker Report" Available at https://www.hackerone.com/sites/default/files/2018–01/2018_Hacker_Report.pdf

[40] J. H. Han, Y. Jeon, J. Kim, "Security considerations for secure and trustworthy smart home system in the IoT environment", *International Conference on ICT Convergence 2015: Innovations Toward the IoT, 5G, and Smart Media Era (ICTC)*, pp. 1116–1118, 2015.

[41] M. S. Haque, T. Atkison, "An Evolutionary Approach of Attack Graph to Attack Tree Conversion," *International Journal of Computer Network and Information Security*, vol. 9, no. 11, pp. 1, 2017.

[42] J. Hong, D. Kim, "HARMs: Hierarchical attack representation models for network security analysis," *in Proc. of the 10th Australian Information Security Management Conference on SECAU Security Congress (SECAU 2012)*, pp. 74–81, 2012.

[43] J. Hong, and D. Kim, "Performance Analysis of Scalable Attack Representation Models", in *Security and Privacy Protection in Information Processing Systems*, Springer, Berlin, Heidelberg, pp. 330–343, 2013.

[44] J. Hong, D. Kim, "Towards scalable security analysis using multi–layered security models," *J. Netw. Comput. Appl*., vol. 75, pp. 156–168, 2016.

[45] J. B. Hong, D. S. Kim, C. J. Chung, D. Huang, "A survey on the usability and practical applications of graphical security models," *Computer Science Review*, vol. 26, pp. 1–16, 2017.

[46] M. Hussain, A. Al–Haiqi, A. A. Zaidan, B. B. Zaidan, M. L. M. Kiah, N. B. Anuar, M. Abdulnabi, "The rise of keyloggers on smartphones: A survey and insight into motion–based tap inference attacks", *Pervasive and Mobile Computing*, vol. 25, pp. 1–25, 2016.

[47] K. Ingols, M. Chu, R. Lippmann, S. Webster, S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs", in *Proc. of the 2009 Annual Computer Security Applications Conference*, vol. 50, no. 1, pp. 117–126, 2009.

[48] K. Ingols, R. Lippmann, K. Piwowarski, "Practical attack graph generation for network defense," in *Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC 2006)*, IEEE, pp. 121–130, 2006.

[49] ISO/IEC, "Risk management – Risk assessment techniques," ISO/IEC 31010, 2009.

[50] ISO/IEC, "Information technology – Security techniques – Information security management systems – Requirements," ISO/IEC 27001 2nd ed., 2013.

[51] ISO/IEC, "Information technology – Security techniques – Information security risk management," ISO/IEC 27005 2nd ed., 2018.

[52] ISO/IEC, "Risk management – Guidelines," ISO/IEC 31000 2nd ed., 2018.

[53] S. Jajodia, S. Noel, "Topological Vulnerability Analysis", in *Advances in Information Security Series*, vol. 46, Cyber situational awareness, pp. 133–154, Springer, 2010.

[54] S. Jajodia, S. Noel, P. Kalapa, "Cauldron: Mission–centric cyber situational awareness with defense in depth", in *Proc. Of the Military Communications Conference (MILCOM),* Baltimore, MD, USA, pp. 1339–1344, 2011.

[55] S. Jajodia, S. Noel, B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publisher, 2003

[56] S. Jajodia, S. Noel, B. O'Berry, "Topological analysis of network attack vulnerability," in: V. Kumar, J. Srivastava, A. Lazarevic (Eds.), *Managing Cyber Threats, in: Massive Computing*, Vol. 5, Springer US, pp. 247–266, 2005.

[57] A. C. Jose, R. Malekian, "Improving Smart Home Security: Integrating Logical Sensing into Smart Home", *IEEE Sensors Journal*, vol. *17, no,* 13, pp. 4269–4286, 2017.

[58] A. Joshi, R. Lal, T. Finin, A. Joshi, "Extracting Cybersecurity Related Linked Data from Text", *IEEE 7th International Conference on Semantic Computing*, Irvine, CA, pp. 252–259, IEEE, 2013.

[59] K. Kaynar, "A taxonomy for attack graph generation and usage in network security", *Journal of Information Security and applications*, vol. 29, pp. 27–56, 2016.

[60] K. Kaynar, F. Sivrikaya, "Distributed attack graph generation", *IEEE Trans. on Dependable and Secure Computing*, vol. 13, no. 5, pp. 519–532, 2016.

[61] S. Khaitan, S. Raheja, "Finding optimal attack path using attack graphs: A survey," *Int. J. Soft Comput. Eng.,* vol. 1, no. 3, pp. 2231–2307, 2011.

[62] S. Khan, S. Parkinson, "Review into State of the Art of Vulnerability Assessment using Artificial Intelligence", in: S. Parkinson, A. Crampton, R. Hill (Eds.), *Guide to Vulnerability Analysis for Computer Networks and Systems*, Springer, Cham, 2018.

[63] B. Kordy, S. Mauw, S. Radomirović, P. Schweitzer, "Foundations of attack– defense trees," in P. Degano, S. Etalle, J. Guttman (Eds.), *Formal Aspects of Security and Trust*, Lecture Notes in Computer Science, Vol. 6561, Springer, pp. 80–95, 2011.

[64] B. Kordy, L. Piètre–Cambacédès, P. Schweitzer, "DAG–based attack and defense modeling: Don't miss the forest for the attack trees," *Computer science review*, vol. 13, pp. 1–38, 2014.

[65] R. Kumar, M. Stoelinga, "Quantitative security and safety analysis with attack fault trees," in *Proc. of the 18th IEEE International Symposium on High, Assurance Systems Engineering (HASE 2017)*, pp. 25–32, 2017.

[66] H. S. Lallie, K. Debattista, J. Bal, "An Empirical Evaluation of the Effectiveness of Attack Graphs and Fault Trees in Cyber–Attack Perception," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1110–1122, 2018.

[67] C. Lee, L. Zappaterra, K. Choi, H. A. Choi, "Securing smart home: Technologies, security challenges, and security requirements", *2014 IEEE Conference on Communications and Network Security, CNS 2014*, pp. 67–72, 2014.

[68] E. LeMay, M. Ford, K. Keefe, W. Sanders, C. Muehrcke, "Model–based security metrics using adversary view security evaluation (advise)", in *8th International Conference on Quantitative Evaluation of Systems (QEST),* pp. 191–200, 2011.

[69] B. Li, and J. Yu, "Research and application on the smart home based on component technologies and Internet of Things", *Procedia Engineering*, vol. *15*, pp. 2087–2092, 2011.

[70] X. Lu, S.–H. S. Huang, "Malicious Apps May Explore a Smartphone's Vulnerability to Detect One's Activities", in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 787–794, 2017.

[71] S. Man, H. X. Yang, Y. Peng, X. S. Wang, "Design of embedded wireless smart home gateway based on ARM 9", *Jisuanji Yingyong/ Journal of Computer Applications*, vol. 30*,* no. 9, pp. 2541–2544, 2010.

[72] E. LeMay, W. Unkenholz, D. Parks, C. Muehrcke, K. Keefe, W. Sanders, "Adversary–driven state–based system security evaluation," in: Proc. *of the 6th International Workshop on Security Measurements and Metrics (MetriSec 2010),* ACM, New York, NY, USA, pp. 5:1–5:9, 2010.

[73] C. Liang, F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges" *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 358–380, 2015.

[74] R. Lippmann, K. Ingols, "An annotated review of past papers on attack graphs," Technical report, MIT Lincoln Lab, 2005.

[75] Y. Liu, H. Man, "Network vulnerability assessment using Bayesian networks," in: B.V. Dasarathy (Ed.), Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005, Society of Photo–Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 5812, pp. 61–71, 2005.

[76] D. Lopez, O. Pastor, L.J. Garcia Villalba, "Dynamic Risk Assessment in Information Systems : State–of the–art," *6th Int'l Conference on Information Technology – ICIT 2013*, pp. 1–9, 2013.

[77] D. López–Pérez, H. Claussen, L. Ho, "The sector offset configuration concept and its applicability to heterogeneous cellular networks", *IEEE Communications Magazine*, vol. *53, no.* 3, pp. 190–198, 2015.

[78] C. D. Martin, "White Hat, Black Hat: The Ethics of Cybersecurity: Taking the High Road", *ACM Inroads*, vol. 8, no. 1, pp. 33–35, 2017.

[79] C. D. Mcdermott, F. Majdani, A. V. Petrovski, "Botnet Detection in the Internet of Things using Deep Learning Approaches", *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

[80] M. McQueen, W. Boyer, M. Flynn, G. Beitel, "Quantitative Cyber Risk Reduction Estimation Methodology for a Small SCADA control system," *in Proc. of the 39th Annual Hawaii International Conference on System Science (HICSS 2006),* Vol. 9, 2006.

[81] J. Meakins (2018): A zero–sum game: the zero–day market in 2018, *Journal of Cyber Policy*, DOI: 10.1080/23738871.2018.1546883

[82] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, Y. Elovici, "N–BaIoT–Network–based detection of IoT botnet attacks using deep autoencoders", *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[83] P.H. Meland, et al., "Security and Trustworthiness Threats to Composite Services: Taxonomy, Countermeasures, and Research Directions," *Secure and Trustworthy Service Composition*, LNCS, vol. 8900 pp. 10–35, 2014.

[84] S. K. Meredith, B. B. Hilliard, M. B. Kosseifi, *U.S. Patent No. 9,158,890*. Washington, DC: U.S. Patent and Trademark Office, 2015.

[85] E. Miehling, M. Rasouli, and D. Teneketzis, "Optimal Defense Policies for Partially Observable Spreading Processes on Bayesian Attack Graphs," in Proc. *2nd ACM Workshop on Moving Target Defense – MTD 2015*, pp. 67–76, Oct. 2015.

[86] E. Miehling, M. Rasouli, and D. Teneketzis, "A POMDP Approach to the Dynamic Defense of Large–Scale Cyber Networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, Oct. 2018.

[87] T. Mouroutis, A. Lioumpas, "D2.1: Use–cases definition and threat analysis", RERUM FP7 project,

2014.

[88] M. Mrinal, L. Priyanka, M. Saniya, K. Poonam, A. B. Gavali, "Smart home – Automation and security system based on sensing mechanism", *Proceedings of the 2017 2nd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2017*, pp. 1–3, 2017.

[89] J. Muniz, G. McIntyre, and N. AlFardan, *Security Operations Center: Building, Operating, and Maintaining your SOC*, Cisco Press, Oct. 2015.

[90] L. Munoz–Gonzalez, E. C. Lupu, "Bayesian Attack Graphs for Security Risk Assessment", *IST–153 Workshop on Cyber Resilience*, 2017 (available in: http://ceur–ws.org/Vol–2040/paper7.pdf).

[91] L. Munoz–Gonzalez, D. Sgandurra, M. Barrere, and E.C. Lupu, "Exact Inference Techniques for the Analysis of Bayesian Attack Graphs," *IEEE Transactions on Dependable and Secure Computing*, Early Access, 2017.

[92] J. Nilsson, "Vulnerability Scanners", Master Thesis, Dept. of Computer and Systems Sciences, Royal Institute of Technology, 2006.

[93] NIST, "Guide for Conducting Risk Assessments," SP 800–30, NIST, 2002.

[94] NIST, "Technical Guide to Information Security Testing and Assessment (SP 800–115)", NIST, 2008.

[95] NIST, "Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach," SP 800–37 Revision 1, NIST 2010.

[96] NIST, "Managing Information Security Risk: Organization, Mission, and Information System View," SP 800–39, NIST, 2011.

[97] NIST, "Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs," Inter–agency Report 7788, NIST, 2011.

[98] NIST, "Guide for Conducting Risk Assessments," SP 800–30 Revision 1, NIST, 2012.

[99] NIST, "Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2," Interagency Report 7275 Revision 4, NIST, 2012.

[100] NIST, "Security and Privacy Controls for Federal Information Systems and Organizations (SP 800–53, Revision 4)", NIST, 2013.

[101] S. Noel, D. Bodeau, R. McQuaid, "Big–Data Graph Knowledge Bases for Cyber Resilience", 2017.

[102] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O'Hare, K. Prole, "Advances in topological vulnerability analysis," in *Proc. of Cybersecurity Applications Technology Conference for Homeland Security (CATCH 2009)*, pp. 124–129, 2009.

[103] S. Noel, E. Harley, K. Tam, M. Limiero, M. Share, "CyGraph: Graph–based Analytics and Visualization for Cybersecurity", *Handbook of Statistics*, Elsevier, vol. 35, pp. 117–167, 2016.

[104] S. Noel, M. Jacobs, P. Kalapa, S. Jajodia, "Multiple coordinated views for network attack graphs," in *Proc. of IEEE Workshop on Visualization for Computer Security (VizSEC 2005),* pp. 99–106, 2005.

[105] S. Noel, S. Jajodia, B. O'Berry, M. Jacobs, "Efficient minimum–cost network hardening via exploit dependency graphs," in *Proc. of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*, IEEE, 2003, pp. 86–95.

[106] S. Noel, S. Jajodia, B. O'Berry, M. Jacobs, "Efficient minimum–cost network hardening via exploit dependency graphs," in *Proc. of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*, IEEE, pp. 86–95, 2003.

[107] S. Noel, S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," in *Proc. of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC 2004),* ACM, New York, NY, USA, pp. 109–118, 2004.

[108] X. Ou, W. Boyer, M. McQueen, "A scalable approach to attack graph generation," in *Proc. of the 13th ACM Conference on Computer and Communications Security (CCS 2006),* ACM, pp. 336–345, 2006.

[109] X. Ou, S. Govindanajhala, A. Appel, "Mulval: A logic–based network security analyzer", in *Proc. of the 14th USENIX Security Symposium*, pp. 113–128, 2005

[110] V. Pandey, R. Saha, *U.S. Patent No. 9,426,095*. Washington, DC: U.S. Patent and Trademark Office, 2016.

[111] N. Penning, M. Hoffman, J. Nikolai, Y. Wang, "Mobile malware security challenges and cloud–based detection", *in 2014 International Conference on Collaboration Technologies and Systems (CTS), IEEE*, pp. 181–188, 2014.

[112] C. Phillips, L. Swiler, "A graph–based system for network–vulnerability analysis," in *Proc. of the Workshop on New Security Paradigms (NSPW 1998)*, ACM, New York, NY, USA, pp. 71–79, 1998.

[113] L. A. Porta, "4 Ways hackers are infecting Phones with Viruses". Available at: https://www.wandera.com/malware–on–android/ (Accessed: 13 December 2018).

[114] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic Security Risk Management Using Bayesian Attack Graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, Jan/Feb. 2012.

[115] A. O. Prokofiev, Y. S. Smirnova, V. A. Surov, "A Method to Detect Internet of Things Botnets", *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 105–108, 2018.

[116] Rambus, "Smart Home: Threats and Countermeasures", 2018. Available at https://www.rambus.com/iot/smart–home/

[117] J. A. Räsänen, *U.S. Patent No. 9,084,233*. Washington, DC: U.S. Patent and Trademark Office, 2015.

[118] N. Rasmussen, "Cyber Security, Terrorism, and Beyond: Addressing Evolving Threats to the Homeland", Hearing before the Senate Committee on Homeland Security and Governmental Affairs, pp. 1–4, 2014. Available at: https://www.dni.gov/files/NCTC/documents/news_documents/cyber_security_terrorism_and_beyond.pdf

[119] S. U. Rehman, V. Gruhn, "An approach to secure smart homes in cyber–physical systems/Internet–of–Things", *2018 5th International Conference on Software Defined Systems, SDS 2018*, pp. 126–129, 2018.

[120] R. J. Robles, T. H. Kim, "Applications, systems and methods in smart home technology: A Review", *Int. Journal of Advanced Science And Technology*, vol. 15, 2010.

[121] R. Ritchey, B. O''Berry, S. Noel, "Representing TCP/IP connectivity for topological analysis of network security", in *Proc. of 18th Annual Computer Security Applications Conference (ACSAC 2002)*, pp. 25–31, 2002.

[122] M. K. Rogers, "The psyche of cybercriminals: A psycho–Social perspective", in *Cybercrimes: A multidisciplinary analysis*, Springer, pp. 217–235, 2011.

[123] S. Roschke, F. Cheng, R. Schuppenies, C. Meinel, "Towards Unifying Vulnerability Information for Attack Graph Construction", in: P. Samarati, M. Yung, F. Martinelli, C.A. Ardagna (Eds.), *Information Security (ISC 2009)*, Lecture Notes in Computer Science, vol 5735. Springer, Berlin, Heidelberg, 2009.

[124] A. Roy, D. Kim, K. Trivedi, "Cyber security analysis using attack countermeasure trees," In *Proc. of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW 2010),* ACM, New York, NY, USA, 2010, pp. 28:1–28:4, 2010.

[125] A. Roy, N. Memon, A. Ross, "MasterPrint: Exploring the Vulnerability of Partial Fingerprint–Based Authentication Systems", *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 9, pp. 2013–2025, 2017.

[126] R. Sabillon, J. Cano, V. Cavaller, J. Serra, "Cybercrime and Cybercriminals: A Comprehensive Study," *International Journal of Computer Networks and Communications Security*, vol. 4, no. 6, pp. 165–176, 2016. Available at: http://www.ijcncs.org/published/volume4/issue6/p1_4–6.pdf.

[127] SAINT Deliverable 3.5 Analysis of Legal and Illegal Vulnerability Markets and Specification of the Data Acquisition Mechanisms

[128] C. Salter, O. S. Saydjari, B. Schneier, J. Wallner, "Toward a secure system engineering methodology," in *Proc. of the 1998 Workshop on New Security Paradigms (NSPW '98).* Charlottesville, Virginia, United States, pp. 2–10, Sep. 1998.

[129] SANS Institute, "Incident Handler's Handbook," SANS Institute – InfoSec Reading Room, pp. 1–19, 2011.

[130] U. Saxena, J. S. Sodhi, Y. Singh, "Analysis of security attacks in a smart home networks", *Proceedings of the 7th International Conference Confluence 2017 on Cloud Computing, Data Science and Engineering*, pp. 431–436, 2017.

[131] H. Sbai, M. Goldsmith, S. Meftali, J. Happa, "A Survey of Keylogger and Screenlogger Attacks in the Banking Sector and Countermeasures to Them", in *Proc. of 10th International Symposium CSS*, pp. 29–31, 2018.

[132] M. Schiefer, "Smart Home Definition and Security Threats", *Proceedings – 9th International Conference on IT Security Incident Management and IT Forensics, IMF 2015*, pp. 114–118, 2015.

[133] B. Schneier, *Secrets and lies: Digital security in a networked world*, John Wiley and Sons Inc., 2000.

[134] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24., no.12, pp. 21–29, 1999.

[135] Y. Seralathan, T. T. Oh, S. Jadhav, J. Myers, J. P. Jeong, Y. H. Kim, J. N. Kim, "IoT security vulnerability: A case study of a Web camera", in *2018 20th International Conference on Advanced Communication Technology (ICACT),* IEEE, pp. 172–177, 2018.

[136] M. Shariqsuhail, G. Viswanathareddy, G. Rambabu, C. V. R. Dharmasavarni, V. K. Mittal, "Multi–functional secured smart home", in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI 2016)*, pp. 2629–2634, 2016.

[137] S. U. N. Shengtao, *U.S. Patent No. 9,178,795*. Washington, DC: U.S. Patent and Trademark Office, 2015.

[138] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated generation and analysis of attack graphs," in *Proc. of IEEE Symposium on Security and Privacy (S&P 2002),* IEEE, pp. 273–284, 2002.

[139] S. Soltani, S. A. H. Seno, M. Nezhadkamali, R. Budiarto, "A Survey On Real World Botnets And Detection Mechanisms", *International Journal of Information and Network Security*, vol. 3, no. 2, pp. 116–127, 2014.

[140] T. Sorell, "Human Rights and Hacktivism: The Cases of Wikileaks and Anonymous,", *Journal of Human Rights Practice*, Oxford University Press, vol. 7, no. 3, pp. 391–410, 2015.

[141] V. Sundar, "3 Ways to Prevent Zero–Day Attacks". Available at: https://www.indusface.com/blog/prevent–zero–day–attacks/ (Accessed: 13 December 2018).

[142] Symantec, "Zero–day vulnerability: What it is, and how it works". Available at: https://us.norton.com/internetsecurity–emerging–threats–how–do–zero–day–vulnerabilities–work–30sectech.html (Accessed: 13 December 2018).

[143] S. Tanwar, P. Patel, K. Patel, S. Tyagi, N. Kumar, M. S. Obaidat, "An advanced Internet of Thing based

Security Alert System for Smart Home". *IEEE CITS 2017 – 2017 International Conference on Computer, Information and Telecommunication Systems*, pp. 25–29, 2017.

[144] M. Thomas, "Survey in Smart Grid and Smart Home Security: Issues, Challenges and Countermeasures", *Health Estate*, vol. 56, no. 8, pp. 24–25, 2002.

[145] N. Tippenhauer, W. Temple, A. Hoa Vu, B. Chen, D. Nicol, Z. Kalbarczyk, W. Sanders, "Automatic generation of security argument graphs," in: *Proc. of the 20th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2014),* pp. 33–42, 2014.

[146] K. Tsipenyuk, B. Chess, G. McGraw, "Seven pernicious kingdoms: a taxonomy of software security errors", in *IEEE Security & Privacy*, vol. 3, no. 6, pp. 81–84, Nov.–Dec. 2005.

[147] R. Unuchek, "Mobile malware evolution 2017", 2018. Available at: https://securelist.com/mobile–malware–review–2017/84139/ (Accessed: 12 December 2018).

[148] M. Urbanska, M. Roberts, I. Ray, A. Howe, and Z. Byrne, "Accepting the inevitable: Factoring the user into home computer security," in Proc. *3rd ACM Conference on Data and Application Security and Privacy*, San Antonio, TX, USA, Feb. 2013.

[149] D. Waltermire and B. Cheikes, "Forming Common Platform Enumeration (CPE) Names from Software Identification (SWID) Tags", NISTIR 8085 (Draft), National Institute of Standards and Technology, Gaithersburg, Maryland, 2015.

[150] D. Waltermire, C. Schmidt, K. Scarfone and N. Ziring, "Specification for the Extensible Configuration Checklist Description Format (XCCDF) Version 1.2", NIST Interagency Report 7275 Revision 4, https://csrc.nist.gov/CSRC/media/Publications/nistir/7275/rev–4/final/documents/nistir–7275r4_updated–march–2012_clean.pdf

[151] T. Warren, "41 percent of Android phones are vulnerable to 'devastating' Wi–Fi attack", 2017. Available at: https://www.theverge.com/2017/10/16/16481252/wi–fi–hack–attack–android–wpa–2–details (Accessed: 14 December 2018).

[152] S. Weerawardhana, S. Mukherjee, I. Ray, A. Howe, "Automated Extraction of Vulnerability Information for Home Computer Security", in: F. Cuppens, J. Garcia–Alfaro, N. Zincir Heywood, P. Fong (Eds.), *Foundations and Practice of Security (FPS 2014).* Lecture Notes in Computer Science, vol 8930, Springer, Cham, 2015.

[153] J. D. Weiss, "A system security engineering process," in *Proc. of the 14th Annual NCSC/NIST National Computer Security Conference*, pp. 572–581, 1991.

[154] L. Williams, R. Lippmann, K. Ingols, "GARNET: A graphical attack graph and reachability network evaluation tool", in: J. Goodall, G. Conti, K.–L. Ma (Eds.), *Visualization for Computer Security*, Lecture Notes in Computer Science, vol. 5210, Springer, Berlin, Heidelberg, pp. 44–59, 2008.

[155] A. Xie, Z. Cai, C. Tang, J. Hu, Z. Chen, "Evaluating network security with two layer attack graphs," in *Proc. of Annual Computer Security Applications Conference (ACSAC 2009),* pp. 127–136, 2009.

[156] R. Yager, "OWA trees and their role in security modeling using attack trees," *Information Sciences*, vol. 176, no. 20, pp. 2933–2959, 2006.

[157] R. Zhuang, S. Zhang, S. DeLoach, X. Ou, A. Singhal, "Simulation–based approaches to studying effectiveness of moving–target network defense," in: *Proc. of National Symposium on Moving Target Research (MTD 2012),* pp. 1–12, 2012.

[158] S. Zonouz, H. Khurana, W. Sanders, T. Yardley, "RRE: A game–theoretic intrusion response and recovery engine," in *Proc. of IEEE/IFIP International Conference on Dependable Systems Networks (DSN 2009),* pp. 439–448, 2009.