jamk.fi

Integrating Attack Graph Analysis System in Semi-Isolated Network Environment

Jani Vanharanta

Master's thesis
June 2017
Information Technology
Master's Degree Programme in Cyber Security

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Vanharanta, Jani	Master's thesis	June 2017
		Language of publication: English
	Number of pages: 97	Permission for web publication: [x]

Title of publication

Integrating Attack Graph Analysis System in Semi-Isolated Network Environment

Degree programme

Master's Degree Programme in Information Technology

Supervisor(s)

Tero Kokkonen, Jari Hautamäki

Assigned by

-

Abstract

The purpose of the thesis was to evaluate the usability aspects of logic-based attack graph analysis for a conventional Network Operation Centre (NOC) in a semi-isolated network environment.

Operational and technical requirements were established for the attack graph analysis, after which an overlook on the outstanding frameworks, standards and models on vulnerability information dissemination was looked into through past research.

The study introduced an existing concept for performing analysis over potential attack combinations in a networked environment. The attack graph analysis concept included methods and technology for logical deductions on the configuration and vulnerability data of the networked environment, resulting in a graphical visualization of the possible and also of the most probable attack paths. The utility that formed the cornerstone for the attack graph analysis was chosen based on previous research.

The usability of attack graph analysis for the NOC was evaluated using the constructed analysis system in a series of use cases. During the use cases the NOC was tasked to monitor a fictitious multinational joint network containing interconnected command & control systems that were using the internally shared core services.

The results for the use cases showed that a conventional NOC can benefit from a logic-based attack graph analysis system. While the graphs compiled from known vulnerabilities easily became too large to use effectively, the analysis helped NOC to increase the security of the hosts through configuration hardening, which ultimately increased the resiliency of the information systems.

Keywords/tags (subjects)

Attack graphs, Cyber Resiliency, Cyber Security, MulVAL, Prolog, SCAP

Miscellaneous

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Vanharanta, Jani Tapani	Opinnäytetyö, ylempi AMK	Kesäkuu 2017
		Julkaisun kieli: Englanti
	Sivumäärä 97	Verkkojulkaisulupa myönnetty: x

Työn nimi

Integrating Attack Graph Analysis System in Semi-Isolated Network Environment

Tutkinto-ohjelma

Master's Degree Programme in Information Technology

Ohjaaja(t)

Tero Kokkonen, Jari Hautamäki

Toimeksiantajat

-

Tiivistelmä

Tutkimuksen tarkoituksena oli tutkia perinteisen verkko-operaatiokeskuksen valvonta- tai käytönohjaustoimintojen hyötynäkökulmia, kun toimintoja täydennetään loogisen päättelyn metodein muodostetuilla kuvaajilla, jotka mallintavat kohdeympäristön mahdollisia hyökkäyspolkuja.

Tutkimustyön aluksi määritettiin hyökkäyspolkujen analysointikykyyn liittyviä operatiivisia ja teknisiä vaatimuksia, jonka jälkeen katselmoitiin aiemmissa tutkimuksissa esiintyneitä ja olemassa olevia haavoittuvuustiedon jakamiseen liittyviä viitekehyksiä, standardeja ja malleja. Seuraavaksi tutkimuksessa esiteltiin olemassa oleva konsepti, jolla verkottuneissa toimintaympäristöissä tapahtuvien, monivaiheisten hyökkäysten analysointi on mahdollista. Kuvaajiin perustuva analysointimenetelmä piti sisällään verkon loogisesta rakenteesta ja siinä olevien laitteiden laiteasetuksista johdettuja loogisen päättelyn toimintoja. Tutkimuksessa käytetty ohjelmisto valittiin aiempien tutkimuksien perusteella.

Tutkimuksen teknisen konstruktion hyötyjä tutkittiin kuvitteellisen organisaation yhteyteen rakennetuilla tapauksilla, jossa verkko-operaatiokeskuksen tuli valvoa monikansallista, yhteiskäyttöistä toimintaympäristöä palveluineen.

Tuloksena todettiin verkko-operaatiokeskuksen voivan hyötyä hyökkäyspolkujen mallinnuksesta. Tunnetuista haavoittuvuuksista analysoidut kuvaajat osoittautuivat monesti liian suuriksi seurata, mutta analyysia voitiin silti hyödyntää ja kasvattaa tietojärjestelmien kybersietoisuutta laiteasetuksiin tehtävillä parannuksilla.

Avainsanat (asiasanat)

Attack graphs, Hyökkäyspolut, Kybersietoisuus, MulVAL, Prolog, SCAP

Muut tiedot

Contents

Abbreviations

APT	Advanced Persistent Threat
AV	Anti-Virus
C2	Command and Control
CAPEC	Common Attack Pattern Enumeration and Classification
CCE	Common Configuration Enumeration
CCSS	Common Configuration Scoring System
CEE	Common Event Expression
CERT	Computer Emergency Response Team
CLI	Command-Line Interface
CPE	Common Platform Enumeration
CRC2	C2 system via Connection node “R”
CSV	Comma-Separated Values
CVE	Common Vulnerabilities and Exposures

CVRF	Common Vulnerability Reporting Framework
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
CWSS	Common Weakness Scoring System
CybEX	Cybersecurity information EXchange framework
CybOX	Cyber Observable eXpression
DDL	Dataset Description Language
DSA	Designated Security Authority
ELK	Elkstack Logstash Kibana
EPS	Encapsulated PostScript
EW	Electronic Warfare
FEC	Forward Error Correction
FIRST	Forum of Incident Response and Security Teams
GNU GPL	GNU General Public License
GSA	Greenbone Security Assistant
HACL	Host Access Control List
HIDS	Host-based Intrusion Detection System
HMN	Harbinger Mission Network
HTML	HyperText Markup Language
IA	Information Assurance
ICMP	Internet Control Message Protocol
ICT	Information and Communication Technology
IDS	Intrudion Detection System
IEC	International Electrotechnical Commission
IODEF	Incident Object Description Exchange Format
IPS	Intrusion Prevention System
ISE	Information Services Environment
ISMS	Information Systems Security Managment
ISO	International Organization for Standardization
ISSM	Information Systems Security Manager
ISSO	Information Systems Security Officer
KVM	Kernel-based Virtual Machine
MAC	Medium Access Control

MAEC	Malware Attribute Enumeration and Characterization
MSM	Making Security Measurable
MulVAL	Multihost, Multistage Vulnerability Analysis Language
NCSA	National Communications Security Authority
NIST	National Institute of Standards and Technology
NOC	Network Operations Centre
NVD	National Vulnerability Database
NVT	Network Vulnerability Test
OCIL	Open Checklist Interactive Language
OpenIOC	Open Indicators of Compromise
OpenVAS	Open Vulnerability Assessment System
OSSIM	Open Source SIEM
OVAL	Open Vulnerability and Assessment Language
PCAP	Packet CAPture
PDF	Portable Document Format
Prolog	PROgrammation en LOGique
RADIUS	Remote Authentication Dial-In User Service
RID	Real-time Inter-network Defense
RID-T	Real-time Inter-network Defense Messages Transport
SA	Situational Awareness
SCAP	Security Content Automation Protocol
SE	Security Engineer
SIEM	Security Information and Event Management
SOC	Security Operations Centre
SSG	SCAP Security Guide
SSL	Secure Socket Layer
STIG	Security Technical Implementation Guides
STIX	Structured Threat Information Expression
SWID	Software Identification
USM	Unified Security Management
VM	Virtual Machine
XCCDF	eXtensible Configuration Checklist Definition Format
XML	eXtensible Markup Language

Figures

Tables

1 Introduction

1.1 Background

In the modern and digital world of today, businesses, organizations and governments increasingly rely on networked information systems to produce, exchange and store different types of business- or- otherwise-critical information. During the digital age organizations have ever increasingly brought their businesses online to utilize the landscape of eCommerce to their best advantage.

Information systems in which the digital information is being processed sometimes present complex and highly sophisticated technology. The information itself, all the combinations of 0s and 1s (and in the future their quantum superpositions) in the storage drives, network attached storages, databases and distributed cloud services, may be important for some and critical for others. For commercial businesses, the importance may be established through competitiveness, market value and patented innovations. A radar manufacturer, on the other hand, may use completely different criteria of, for example, integrated circuitry details of a frequency-hopping EW resistant radar.

In the meantime, governments have also started to make their services available to the public online. Such eServices have been launched by several agencies enabling, e.g. electronic voting, social security services, electronic passport applications, and vehicle registration service.

There is even a possibility to become a virtual e-resident of Estonia, allowing for establishing and starting of real businesses that fully integrate to the Estonian electronic services of the digital infrastructure (Hammersley, 2015).

Along with the emerging eServices, responsibilities have also grown. The users of the services need to be assured of their security. This is a common denominator for the service providers in business-to-business, consumer-to-business, as well as government-to-public operations. Service providers in each operational area are required to assure the security of their environment, the part of the whole digital

space, in which all the electronic information is being accessed, processed and stored through computers, countless servers and myriad of information systems.

Malicious software, *malware*, that is nowadays being discovered, has evolved significantly from the ones that were the first to utilize computer networks as their propagation paths. One of the first, even if not intentionally *malicious*, was the *Morris Worm*, discovered in 1988 and named after its creator Robert Morris. The worm was designed to propagate the network and to demonstrate the inadequacies of computer network security measures (Spafford, 1988; Morris, 1988). Even if the malware today has several similarities to the persistent, obfuscating and encrypting code of the *Morris Worm*, the most advanced of them are being used in a completely different types of campaigns with far more serious motivation in their background, as seen in several report and trend publications (FireEye APT28, APT29, APT30; Mandiant APT1, M-Trends; F-Secure Threat Report 2015).

As operations have become increasingly reliant on networks, securing them has also become more demanding, and resource consuming for the network and system engineers. Even disregarding the fact that the number of vulnerabilities within a network easily gets multiplied by the software combinations of the devices within, the rate at which they are reported (Figure 1) can be overwhelming. Assuring the security of the information systems may prove difficult to achieve, especially for small-to-medium-sized businesses and their perhaps cost-effectively balanced network security or administrator teams- but also for larger companies that may have outsourced their whole ICT functions.

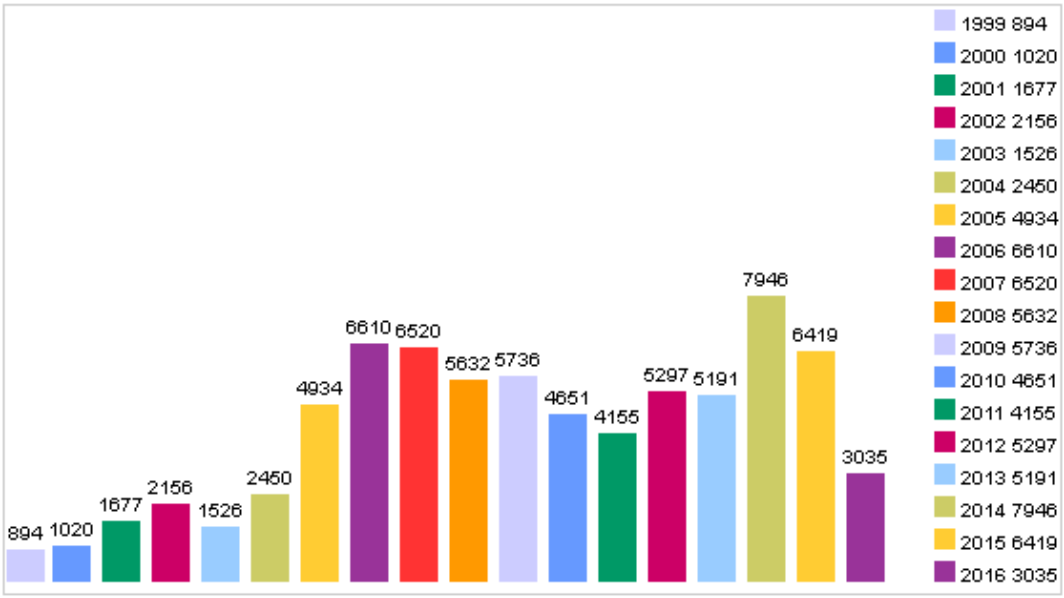

Figure 1. Vulnerabilities by Year (www.cvedetails.com, referred 21.6.2016)

In an information ecosystem where different operating systems and products from multiple device and software vendors with various software versions and subversions coexist, vulnerabilities and their possible combinations can present significant risks to the environment. Vulnerabilities may exist in the operating systems, in the firmware software, in the application software and in their library objects already when they are released, only to be discovered later. New vulnerabilities can also make their ways into the environment during new software releases, software upgrades and even security updates. When exploitable, they can cause serious damage to the assets in the environment and endanger the confidentiality, integrity and availability of the information.

Single vulnerabilities are easily seen as individual threats, which by themselves, may seem to present only a minor threat, while they may in fact be involved in a series of actions through a combination of multiple vulnerabilities in a campaign against the organizations assets, ultimately putting the entire environment at risk.

In addition to vulnerabilities, also sub-optimal configuration or configuration errors can expose the environment to threat agents and eventually lead to realization of risks. Configuration changes often link to scheduled maintenance or incident resolving- but can also occur during services, business operations, or just network expansion. Without a thoroughly implemented and maintained configuration or

asset management, how can one determine whether there have been any such changes in the software base or in the configurations that could introduce threats into the environment?

Wang, Jajodia, Singhal & Noel (2010) and Ou & Singhal (2011) underline that one cannot improve what one cannot measure. During the last decade, a considerable amount of research has been conducted on measuring network security. Ou, Boyer & McQueen (2006) presented a logical approach to represent and generate attack graphs, designed to illustrate the logical dependencies among attack goals and configuration information. In their research they established the necessity of considering multi-stage and multi-host attacks. Ou & Singhal (2011) suggest that the overall security of a network environment cannot be determined by simply listing the number of vulnerabilities they contain; instead, they presented a methodology for a composition of multiple vulnerabilities being modelled using probabilistic attack graphs, ultimately showing all paths of attacks that when combined, will enable multistage network attacks.

Situational awareness is at focal point in protecting networked environments. In that respect attack graphs can be seen as a source for ample information for the network and security administrators. Information security automation has brought different kinds of data streams and analytical functions in great numbers for the network operators to use. But instead of assistance, do they in fact cause disorientation through information overflow? Can the security or network engineer handle all the information flows and effectively correlate the security related events?

Could the analysis itself be automated in such a way that it would support the engineer by providing readily prioritized and weighed suggestions or recommendations as to how to react and what actions they should take to protect their environment most effectively?

1.2 Research Objective

The objective of this thesis consisted of two parts: Firstly, to help solve the problem of measuring and analyzing overall security of a semi-isolated network environment in which several information systems coexist, most of which are interconnected, and

where the vulnerability information may be few and scattered and cover only individual assets. Secondly, to assess whether an attack graph analysis system could be integrated into a Network Operations Center and how such integration would improve the ability to defend against an attack.

In support of the primary objective, the research aimed to provide technical assistance for existing risk assessment by enabling cyclic attack graph analysis that could become a part of the process for managing software upgrades and configuration changes in the operational environment.

Supporting the secondary objective, this research sought to find an answer to the question *is a logic-based attack graph analysis system suitable for Network Operations Center's (NOC) use*? The usability was assessed through a fictitious use case involving an Information Systems Security Manager (ISSM) and an Information Systems Security Officer (ISSO), with limited initial situational awareness which was correlated with new information in the environment on which the ISSM and ISSO could (re)act.

The work conducted in this thesis is expected to contribute to the risk management process of the organization, to provide a high-level perspective of a functional SIEM system complemented with attack-graph-analysis capabilities, and to initiate course of action algorithm development focusing on information systems' autonomous defenses.

1.3 Research Method

The method of research selected for this thesis is constructive research. Constructive research was chosen regardless of the risk of lacking objectivity.

Constructive research is a method for producing a construction intended to help solve real-world problems or part of the problems. Realizations of the construction, artifacts solve a domain specific problem in order to build knowledge on how the problem can be solved (or understood, explained or modelled) in principle. Artifacts such as models, plans, organizational charts, information system designs, algorithms and software development methods are typical constructs used in research and

engineering. Characteristic to them, they are invented and developed, not discovered (Dodig-Crnkovic, G. 2010, Lukka, K. (Internet, N.D.).

Lukka (2000) also suggests that the core characteristics for a constructive research include that it

- focuses on real-world problems felt relevant to be solved in practice
- produces an innovative construction meant to solve the initial real-world problem
- includes an attempt for implementing the developed construction and thereby a test for its practical applicability
- implies a close involvement and co-operation between the researcher and practitioners in a team-like manner, in which experimental learning is expected to take place
- is explicitly linked to prior theoretical knowledge, and
- pays particular attention to reflecting the empirical findings back to theory

Similarly, Labro & Tuomela (2003) present seven crucial steps in the constructive research approach as illustrated in segments of three phases in Figure 2: 1) find a practically relevant and theoretically interesting problem; 2) examine the potential for long-term co-operation with the organization; 3) obtain a comprehensive understanding of the topic; 4) innovate and construct a theoretically grounded solution; 5) implement the solution and test whether it works in practice; 6) examine the scope of the construct's applicability and; 7) show the theoretical connections and the research contribution of the construction.

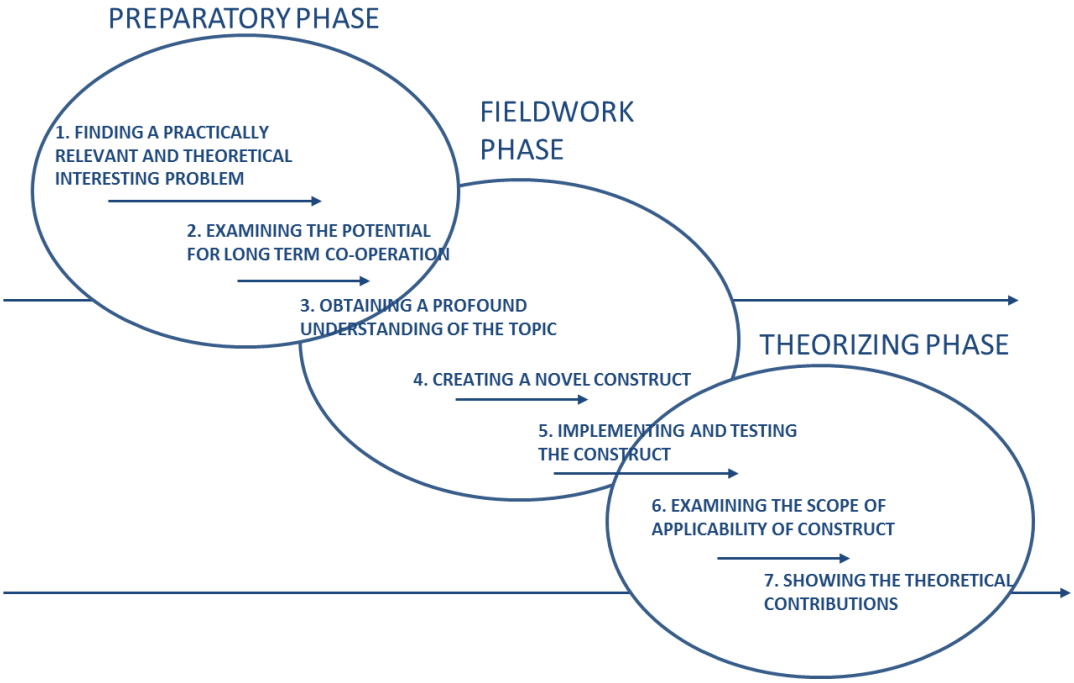

Figure 2. Crucial Steps in Constructive Research Approach by Labro & Tuomela (2013)

For this research, the recognized real-world problem is presented in the previous chapters 1.1 and 1.2: Improve computer network defense performance for cyber operations decision makers and NOC by providing analysis capability for multistage computer network attacks and course of action guidance. The potential for a long term co-operation with the organization fundamentally exists, with the involvement of cyber operations decision makers and the NOC. In this research the construction application is the integration – technical implementation – of the attack graph analysis system in a semi-isolated network environment where NOC capability already exists. Significant theoretical contribution is not expected from this research.

1.4 Thesis Structure

The first chapter of the thesis is an introduction, describing the background, method and the objectives for the thesis. The second chapter presents the functional and technical requirements for attack graph analysis.

The third chapter introduces the attack graph concept: the ability to perform analysis and risk assessment of multistage attacks in a network-centric environment, and how and based on what information the actual attack graphs are generated.

Vulnerability information and their common sources and respective standards and specifications are introduced, as well as how the attack graph system is utilizing the vulnerability information, and what metrics and scoring systems are involved in the risk quantification. Additionally in chapter three, some of the vulnerability information sharing models are presented through previous work and studies on vulnerability data structuration and sharing standardization.

The third chapter will also briefly introduce some of the most common vulnerability scanners, one of which was used extensively for the attack graph engine included in this research.

The logic behind the attack graph analysis engine's deduction process is presented through logic programming paradigm. Three of the most important logic programming components in the attack graph analysis framework are looked into in more detail.

Concluding chapter three, the algorithms for the attack graph analysis and graph construction are introduced. The attack graph architecture is presented and the attack graph compilation is demonstrated with a practical example.

The fourth chapter of the thesis presents the construction of the attack graph analysis: how the vulnerability assessment data of the test network is compiled for graphical presentation and, how the grouping algorithm and the quantitative risk assessment change the attack graph abstraction.

The attack graph concept and its construction are tested in chapter five with a fictitious joint mission network use case.

Chapter six presents the conclusions on the usability of the construction against the preconditions and research method together with some plausible paths for future research.

2 Requirements for Attack Graph Analysis

Attribution of the malefactors and analysis of their motivations would be essential in effective cyber defense, however for the brevity of this research, they were excluded and the actors were assumed as state-sponsored malefactors. The environment in this research was treated as semi-isolated as it was not directly connected to the Internet or any network other than those in the Harbinger Mission Network (HMN) use case.

2.1 Operational requirements

To allow the network operators to perform analysis over any data, a concept needs to be defined. Ultimately the network or cyber operations decision makers must be provided with enough valuable data in such a way that the operations can be run safely enough and long enough relative to the operational need. The sheer amount of data sources and data types necessitate that the data will have to be formatted to and presented in human understandable form, to help decision makers understand the possibly complex overall situation.

The decision makers also need to understand what kind of decisions they are required to make and what kind of actions exist in the "*playbook*" with which the NOC is able to protect the respective assets in the environment. The analysis and implementation of such decision support is, however, outside the scope of this thesis. Therefore the decision making process was simplified and was considered well prepared and temporally effective.

The ability to construct a practical presentation of the possible attacks requires that they can be modelled. The modelling will have to take into account the existence of single or multiple vulnerabilities and also device and software configurations such that present risks to the environment by running configurations that are against best practice or vendor recommendations or otherwise sub-optimal.

The underlying risk management process requires that the vulnerability data for the operational environment will have to be updated on a regular basis, despite the fact that the operation takes place in a semi-isolated network environment.

The aim with the solution in this research is to enable safe operation of the services within the network environment, to help assess the risks against the assets, to provide sufficient protection against targeted cyberattacks, to assist in mitigating and stopping multi-staged attacks, to provide protection against zero-day attacks, and finally to enable autonomous protection mechanisms to suppress the attackers and wear their resources.

In this research, the operational requirements are derived from the following three tenets from Jacobson (2013):

1. predict plausible impact of cyberattack situations before they occur
2. survive through adaptation and degradation during the attacks
3. recover operational capacities after the attack

Through attack graph analysis the aim was to predict the attacker's probable steps in multistage attacks and to enable the planning of manual and automated parallelized or serialized responses that help sustain the attack and strain the attackers' resources.

2.1.1 Situational Awareness

To be able to successfully operate in the cyberspace, situational awareness must be effectively utilized to enable decision makers lead and come up with timely decisions (Dressler et al. 2014).

Situational awareness seems seldom used in conjunction with the words cyber or cyber security. While a great amount research has been conducted on attack graphs, not many seem to highlight the importance of the situational awareness angle to it. In their research on operational data classes Dressler et al. (2014) referred to battlespace awareness as a closest candidate for the definition.

Conti et al. (2013), referred to U.S. Military Doctrine in their definition situational awareness as "*the requisite current and predictive knowledge of the environment upon which operations depend…*"

Conventional solutions such as intrusion detection and prevention systems and anti-virus products, all of which build to situational awareness, but by themselves do not suffice especially with isolated networked environments and state-sponsored actors. Such conclusion is established by Hutchins et al. (2010), in their paper on adversary campaigns and intrusion kill chains. They underline that solutions relying on conventional methods fail due to their fundamentally false assumptions such that a response takes place only after a compromise and that the compromise was a result of a problem that is easily fixable.

In their work, Hutchins et al. (2010) claim that regardless of the positive development in information management tools that have resulted in, e.g. best practices, hardening and rapid patch deployment, the state-sponsored malefactors have still been able to continually demonstrate system compromise capabilities through advanced tools, customized malware and zero-day exploits.

To effectively build to the resilience of the protected assets in the thesis network, the decision makers must have at their disposal at a minimum, a graphical presentation of the outstanding vulnerabilities discovered in the protected assets and also of the interdependencies of the assets in the infrastructure. In the scope of the thesis, this requirement represented the required level of situational awareness for the correlation, and fusion of the analysis data.

2.1.2 Impact Mitigation

Established by Hutchins et al. (2010), the methodology that uses knowledge-based conventional solutions would not be efficient against multi-staged APT attacks in which the malefactors continually change their actions according to the environment.

The attack graph analysis used in this thesis will need to, to an extent, assist the NOC for predicting the most probable steps in a multistage attack in the given environment. Optimally, the construction would enable for the dynamic and continual changes in the defended environment in order to increase the attack cost

for the malefactors and to increase the resiliency of the environment against repeatedly used techniques.

The solution in this research will need to support for the adaptation of the environment to enable for instance the following mitigative abilities:

1. rerouting and slowing down traffic
2. straining the attacker's resources
3. shutting down, moving, or creating new hosts for taking over processes

Additionally, the ability to investigate the incidents to be able to increase and enhance the knowledge in our knowledge-based counteraction engines is desirable. Such ability would mean, for instance, creating and altering signatures for the IDS-sensors of the SIEM system.

The solution will also need to consider the so-called zero-day vulnerabilities, such that are currently unknown to any hardware or software vendors and not yet "*fixable*" with a conventional software upgrade or patching. This method will increase the resiliency before the attacks, and enable in-time planning for counteractions in the thesis test network.

According to Hutchins et al. (2010), the ability to revisit the attacks and reconstruct intrusions is particularly useful. Being able to recognize patterns or signatures of unorthodox and advanced methods could prevent their reuse and would likely increase the required cost of the malefactors' campaigns.

2.2 Technical Requirements

2.2.1 Vulnerability, Software and Configuation Information

The capability to predict multistage attacks such that utilize known exploits requires an ability to produce a comprehensive list of all the vulnerabilities that exist in the thesis network environment. In addition, the ability is needed to assess and interpret every possible combination of individual steps that allow for a multistage attack to take place.

With regard to the vulnerability information, the categorized information sources and their standardized formats, that are currently considered the most commonly used in cyber and vulnerability information sharing, will be utilized.

Software and configuration information will be established by using vulnerability scanners, through compliance audits and via an asset database, the latter of which will be an internal component of the SIEM system.

2.2.2 Host Access Lists

A comprehensive access list of all the hosts in the internal as well as the DMZ and perimeter networks will need to be created for the logic engine. Without such a list, the multistage attack graph would be imprecise and could lead to ineffective reactions to the attack(s) and making the preplanning of the counteractions difficult.

2.2.3 Intrusion Detection

Along with the SIEM system, a knowledge based IDS will be deployed, consisting of one network sensor, with the option to produce new sensors through an automated virtual guest machine response mechanism. The IDS will be installed from the OSSIM installation media, and the IDS will report possible events and suspected malicious activities to the SIEM system for the NOC to further analyze. The patterns and signatures for malicious activity will be provided by the Emerging Threat Open ruleset.

2.2.4 Counteractions

Manual and automated counteractions will be made available through the management network and the HIDS component which is integrated into the SIEM system. The HIDS agent will be installed to the supervised hosts where applicable. Active network devices such as routers and switches, where the installation of HIDS agent is not possible, will be supervised from within the SIEM system as agentless hosts.

Counteraction capabilities include, but are not limited to:

1. reactive firewall rule addition(s) on the core firewall and Linux-based hosts through orchestration

2. proactive and reactive full packet capture for analysis and forensic investigation
3. scripted known-safe-configuration revertations through orchestration
4. manual or semi-automated reallocation of a VM guest into a quarantine network based on a triggered event
5. manual or semi-automated deployment of the IDS sensor to the subnet in which the event occured
6. relocation of the IDS sensor to the network subnet into which (based on the probability analysis) the attack would likely be directed

The counteractions can be triggered based on network events, and also manually through the HIDS component or by scripted orchestration through the management network.

3 Attack Graphs

3.1 Multistage attacks

An attack that consists of several subsequent steps and utilizes exploits to known vulnerabilities can be modelled with a tool that can present all the steps in the attack and such that also has information on the assets in the network, their interdependencies as well as information on their individual vulnerabilities. Gallon & Bascou (2011) introduce attack graphs as visualizations of the attack model derivations, of every possible scenarios in which an attacker can achieve certain goals in the network. Similarly, Kotenko & Chechulin (2013) define attack graph as a graph representing every possible sequence that lead the attackers to their goals. As they mention such cyberattack modelling as one of the promising approaches, they also underline the computational complexity of the graphs and challenges in their utilization in near-real-time systems.

To be able to operate in cyberspace, organizations have to have a certain level of awareness of the operating environment (Dressler et al. 2014). The required level of broadness and depth varies greatly depending on the size and trade of the business. However, without access to databases and knowledge base of vulnerabilities, or the

respective collaborative networks, the organization may have limited or a completely false view of their situation and of the security posture of their assets.

For an organization that does not have resources for analysis, processing and dissemination of such vulnerability related information, it could be highly beneficial to participate in programs for vulnerability information sharing frameworks and platforms.

3.2 Vulnerability Information Sharing

Tosh et al. (2015) underlined an important aspect regarding companies that after a discovered compromise, in fear of an image hit, negative publicity, will likely refrain from disseminating or publishing information of the particular incident. That information could be of great value to other companies that share similarly configured infrastructure, e.g. the same cloud infrastructure. Refrainment from disclosing such information may of course derive from outstanding laws, especially in case of agencies and governmental institutes, and service providers for that sector.

Information security and cyber-related threat information sharing has been the topic in several studies and work projects in the recent past. Kamhoua et al. (2015), for instance, applied game theory to investigate when multiple self-interested companies could invest in vulnerability discovery and sharing their threat-related information. Game theory was also used in Tosh et al. (2015) paper, where they formulated a non-cooperative cybersecurity information sharing game to guide the companies to independently decide whether or not to share information.

In the absence of effective publishing and sharing mechanisms, Gadelrab & Ghorbani (2012) proposed an approach to express network and security dataset metadata using a Dataset Description Language (DDL). According to their paper, the proof-of-concept prototype implementation produced XML output, such that could be integrated with Security Content Automation Protocol (SCAP) tools.

Since the work of Gadelrab & Ghorbani, several standardized and automated information expression and sharing standardization and structuration attempts have emerged. In their work on semantic ontologies for cyber threat sharing standards, Asgarli & Burger (2016) mention STIX, IODEF and OpenIOC as examples. MITRE’s

Structured Threat Information eXpression , STIX™ according to Asgarli & Burger's is considered the most extensive standard, having definitions for cyber observables, indicators, incidents, exploit targets, attack methodologies, courses of action, threat actors and campaigns (Asgarli & Burger, 2016).

In a study on information sharing models Hernandez-Ardieta et al. argued that effective policies for near-real time information sharing must rely on firstly, on development of formal models estimating subjective value of the shared information and secondly, on identifying the trust and reputation models that consider dynamic behavior and changing factors of the information sharing community. In their work they also wrote about efforts on categorizing and standardizing data formats and exchange protocols on information concerning cyber security: the assets and their configurations, threats and tactics utilized by the attackers, as well as indicators of compromise and risk mitigating counter-actions. As an example of such structural model they mention Making Security Measurable™ initiative, by The MITRE Corporation. (Hernandez-Ardieta et al. 2013.)

Architecturally MSM™ includes building blocks such as enumerations of common concepts that need to be shared, languages defining how to find and disseminate such concepts and repositories through which to share the standardized content in a machine-consumable form (Introduction to Making Security Measurable, The MITRE Corporation, 2016).

Some of the most notable MSM enumerations include Common Vulnerabilities and Exposures (CVE ®) detailing standard identifiers for publicly known vulnerabilities, Common Platform Enumeration (CPE) with standard identifiers for platforms, operating systems and software packages, and Common Weakness Enumeration (CWE™) identifying software weaknesses in architecture, design or implementation that lead to vulnerabilities (Introduction to MSM).

Languages and formats in MSM include, among others, Open Vulnerability and Assessment Language (OVAL ®), a language to write XML-based tests on current state of assets and for displaying the results, and Common Vulnerability Scoring System (CVSS), a methodology for disseminating vulnerability related risks and risk measurements (Introduction to MSM).

MSM repositories for sharing the standardized content in machine-readable format, such that are publicly available, include (U.S.) National Vulnerability Database (NVD), a vulnerability database based on CVE that integrates all the publicly available resources and references, and the OVAL repository for OVAL vulnerability, compliance, inventory and patch definitions (Introduction to MSM).

In their work Hernandez-Ardieta et al. (2013) presented a grouping of the current standards into processes and their mapping into six knowledge areas [A]sset definition (inventory), [C]onfiguration guidance (analysis), [V]ulnerability alerts (analysis), [T]hreat alerts (analysis), [I]ndicators (intrusion detection), and [R]eport (management), (Hernandez-Ardieta et al. 2013).

The MSM standards and their respective knowledge area mapping as in Hernandez-Ardieta (2013) are shown in Table 1.

Table 1. MSM Standards and Knowledge Areas. Quoted from Hernandez-Ardieta et al. (2013).

	CPE	OVAL	SWID	XCCDF	CCE	OCIL	CCSS	CVE	CWE	CVSS	CAPEC	CVRF	MAEC	CybOX	IndEX	STIX	IODEF	CEE	RID	RID-T	CYBEX	CWSS
A	•	•	•																			
C		•		•	•	•	•															
V		•						•	•	•		•										
T								•	•	•	•		•	•	•	•	•	•	•	•		
I	•							•					•	•	•	•	•	•	•	•		
R	•	•			•			•	•	•			•		•	•	•		•	•	•	•

The components in the construction in this research will utilize some of the aforementioned standards such as OVAL, CVSS and XCCDF all of which are included in the knowledge areas listed by Hernandez-Ardieta et al. (2013).

3.2.1 OVAL®

Open Vulnerability Assessment Language is an international, information security, community standard for promoting open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security

tools and services. OVAL includes a language used to encode system detail, and an assortment of content repositories help throughout the community (About OVAL).

The OVAL community has developed three schemas, written in XML, to serve as the framework and vocabulary of the OVAL language. The schemas correspond to the three steps of the assessment process: System Characteristics schema for step 1) representing configuration information of systems for testing; an OVAL Definition schema for step 2) analyzing the system for the presence of the machine state (vulnerability, configuration, patch state); and an OVAL Results schema for step 3) reporting the result of the assessment (About OVAL).

3.2.2 CVSS

Common Vulnerability Scoring System is a specific scoring system designed as an open framework and standardized method for rating and disseminating vulnerability characteristics and their impacts for ICT components. CVSS assists organizations in prioritizing and channeling resources needed for handling security incidents. The quantitative model of the CVSS ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the scores. Thus, CVSS is well suited as a standard measurement system for industries, organizations, and governments that need accurate and consistent vulnerability impact scores (CVSS, FIRST.org.)

CVSS assessment consists of three metric groups; Base, Temporal and Environmental, each with its own set of metrics, as shown in Figure 3. Each group produces a numeric score ranging from 0 to 10, and a *Vector*, a textual representation of the values that were used to derive the score (CVSS, FIRST.org).

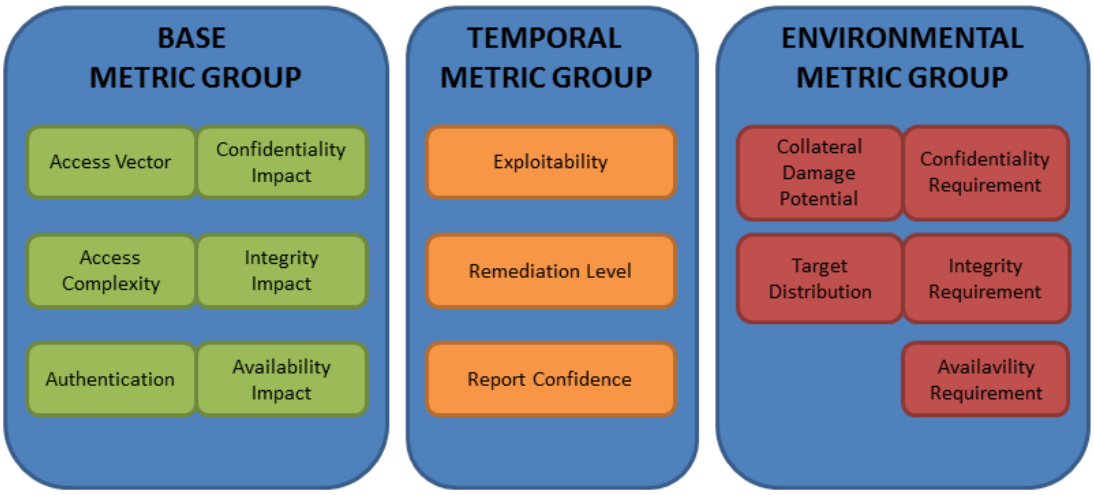

Figure 3. CVSS Metric Groups by Mell et al. (2007)

The metric groups are described as follows (CVSS, FIRST.org):

> ***Base**: represents the intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environments.*
>
> ***Temporal**: represents the characteristics of a vulnerability that change over time but not among user environments.*
>
> ***Environmental**: represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment*

The base metric group defines the characteristics of the vulnerabilities that remain unchanged over time and over different environments. In Figure 3 the Access Vector, Access Complexity and Authentication metrics define, the vectors from which the vulnerabilities can be exploited, how difficult their exploitations are and whether or not authentication is needed for them to succeed. The impact metrics for Confidentiality, Integrity and Availability on the other hand, independently define how the successful exploits can affect the confidentiality, integrity and availability of the assets (CVSS, FIRST.org).

3.2.3 CVE

Common Vulnerabilities and Exposures (CVE®) is a dictionary of common identifiers for publicly known information security vulnerabilities. CVE was launched in 1999 when most information security tools used their own databases and naming conventions for vulnerability related information, and distinguishing problem descriptions from another, potentially resulting in multiple referrals to the same problem. CVE started producing standardized identifiers (CVE identifiers) for reference points for vulnerability data exchange. CVE identifiers also provide a baseline for evaluating the coverage of tools and services so that users can determine which tools are most effective and appropriate for their organization's needs. CVE is currently the industry standard for vulnerability and exposure names. (CVE, About CVE.)

3.2.4 NVD

NIST Computer Security Division's National Vulnerability Database (NVD) provides a framework for disseminating the vulnerability characteristics and their potential impacts on ICT infrastructure (NVD Home, 2015).

> *NVD is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. NVD includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics. (NVD Home, 2015.)*

NVD supports the CVSS v2 standard for all CVE vulnerabilities, although it provides only the CVSS base scores for vulnerabilities. NVD does, however, provide CVSS calculators for adding temporal scores, and to some extent measuring environmental scores to reflect an impact of a vulnerability to a specific organization's environment (NVD Home, 2015).

3.2.5 SCAP

Security Content Automation Protocol is a specification for expressing and handling security-related data in a standardized way. Using many individual specifications, SCAP automates continuous configuration monitoring and vulnerability management. The eXtensible Configuration Checklist Description Format, XCCDF, that is a part of SCAP specification, is a language for writing security checklist and benchmarks for use in compliance checklists and security policies (Security Content Automation Protocol, SCAP).

The currently effective version of SCAP (specification version 1.0) contains XCCDF and OVAL languages, CCE, CPE and CVE enumerations and CVSS metrics (SCAP Specification). Currently, most of the SCAP validation products support the version 1.2 of the specification (NVD, Security Content Validation Products).

According to Alsaleh & Al-Shaer (2011), SCAP is a set of interrelated specifications that represent the standard format and nomenclature by which security software communicates information about known software flaws and configurations. In their work, Alsaeh & Al-Shaer (2011) take on SCAP as a standard way for representing and measuring information security within a system. Similarly, Hlyne et al. (2015) introduce SCAP as a suite of specifications that help organizations to automatically assess their network devices, operating systems and applications for their respective security configuration compliance, and to help automate security management tasks.

SCAP validation software utilities such as OpenSCAP allow for system administrators to check configuration settings and examine the system for signs of possible compromise through the use of rules that are based on SCAP standard and specification (OpenSCAP User Manual).

3.3 Vulnerability Assessment

To be able to improve the security or resiliency of the network-centric environment against threats and attacks, the initial status of the environment needs to be established. For enterprise networks, vulnerability assessment is the way to test

whether any discovered and known vulnerabilities exist in their environment, and decide on the level of the risk they introduce. (Gallon & Bascou, 2011.)

Vulnerability assessment is usually determined by combining various factors such as how much effort is needed for a malefactor to reach and exploit a vulnerability, the required mode of authentication in the target system, if they are exploitable locally or via remote access, and the impact on the confidentiality, integrity and availability of the information (Zhang et al. 2011).

For such factors, CVSS scoring provides intrinsic assessments on the fundamental characteristics of the vulnerabilities, such that do not change over time and environments through the metrics of the base group (Gallon & Bascou, 2011).

3.4 Examples of Vulnerability scanners

3.4.1 Nessus

Nessus is a commercial vulnerability scanner by Tenable Network Security, Inc. Nessus is used to discover all the assets on the network and test them for existing vulnerabilities or missing patches. In addition to a non-credentialed, remote scans, Nessus also supports deeper, granular analysis of assets through credentialed scans, and offline auditing for network devices. For configuration and compliance auditing, Nessus utilizes over 450 templates (Nessus Professional).

Newly discovered vulnerability information is transformed into plugins by the Tenable's research staff to enable Nessus to detect them. The plugins are written in the Nessus Attack Scripting Language (NASL), and they contain vulnerability information, a generic set of remediation actions and the algorithm to test for the presence of the security issue. The plugins are provided as streams, Feeds, which are available as a subscription purchase (Nessus Plugins).

Nessus also provides a plugin feed for home users, Nessus® Home, which can be used for scanning an environment up to 16 individual IP addresses. In comparison with Nessus Professional, Nessus® Home lacks the ability to conduct compliance checks or content audits (Nessus ® Home).

According to Daud et al. (2014), the Home Feed release only includes the latest plugins up to the installation date, whereas the Professional release gets the plugins updated continuously as per the outstanding subscription.

In addition to the Nessus Professional's ability to conduct compliance audits through the plugins provided by the paid subscription, Windows and Linux SCAP compliance checks are enabled through the "*SCAP Windows Compliance Checks*" and "*SCAP Linux Compliance Checks*", respectively, provided that the specified policy against which the audit be performed, contains XCCDF or OVAL-formatted SCAP content (Nessus Compliance Checks, Nessus v6 SCAP Assessments).

3.4.2 OpenVAS

Open Vulnerability Assessment System (OpenVAS) is a framework of several services and tools offering a comprehensive and powerful vulnerability scanning and vulnerability management solution. The OpenVAS Scanner service is accompanied with a regularly updated feed of Network Vulnerability Tests (NVTs). All OpenVAS products are Free Software, and the components, most of which are licensed under the GNU General Public License (GNU GPL). (About OpenVAS)

Figure 4 depicts the OpenVAS architecture. OpenVAS CLI and Greenbone Security Assistant are the applications for user interaction with OpenVAS Manager, which is the central service that consolidates plain vulnerability scanning into a full vulnerability management solution. The OpenVAS Scanner is the core of the OpenVAS architecture, in charge of executing the actual Network Vulnerability Tests, which are served via the NVT Feed. (OpenVAS Architecture Overview)

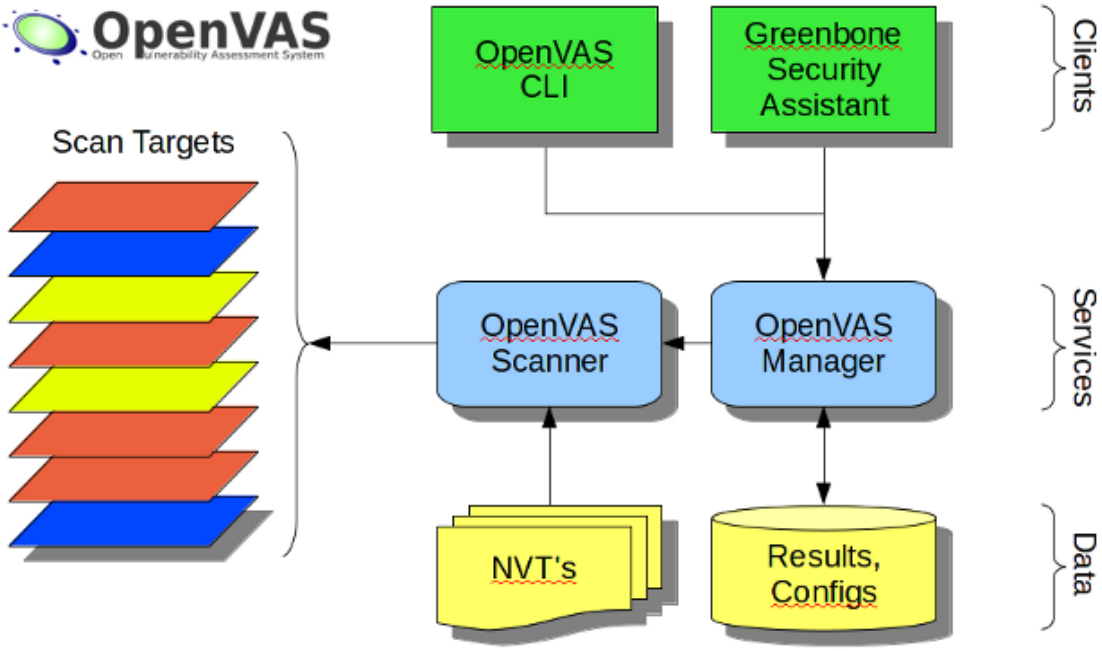

Figure 4. OpenVAS Architecture (About OpenVAS)

OpenVAS is an official participant in OVAL Adoption Program by MITRE (OpenVAS Architecture Overview).

3.4.3 Nexpose

Nexpose is a commercial vulnerability management product by Rapid7. Nexpose provides a dashboard view for managing vulnerabilities, security patches, and analytics and reports in large infrastructures. Nexpose claims to provide live view into vulnerabilities as they happen, and provides remediation and best practices for secure configurations. (Nexpose)

3.4.4 OpenSCAP

OpenSCAP is not particularly a vulnerability scanner. Rather, it is more of a configuration assessment framework. According to their web site SCAP is a project, providing a wide variety of hardening guides and configuration guidelines. It is also an ecosystem providing a collection of open source tools for implementing and enforcing the U.S. NIST Security Content Automation Protocol, SCAP. Vulnerability assessment in OpenSCAP is enabled through an automated software inspection and

security configuration settings check mechanism, looking for signs of compromise by using rules based on standards and specifications (OpenSCAP).

OpenSCAP uses SCAP, processing mainly the XCCDF, which is a standard way of expressing checklist contents and defining security checklists. It also combines with other specifications such as CPE, CCE and OVAL to create SCAP-expressed checklist (OpenSCAP User Manual).

OpenSCAP is able to evaluate both XCCDF benchmarks and OVAL definitions and creating the respective results. The following are two example commands for invoking evaluations for OVAL and SCAP on a Red Hat Enterprise Linux 6 with a sample security policy, and a sample security guide template, respectively:

```
oscap oval eval --results rhva-results-oval.xml –
report oval-report-highside.html Red_Hat-
Enterprise_Linux_6.xml
```

```
oscap xccdf eval --profile
xccdf_org.ssgproject.content_profile_usgcb-rhel6-
server --results-arf arf.xml --report xccdf-report-
highside.html /usr/share/xml/scap/ssg/content/ssg-
rhel6-ds.xml
```

During the testing of OpenSCAP for its applicability for vulnerability assessments it was interesting to make note that OpenSCAP had the ability not only to perform compliance audits based on completely customized benchmarks, but also to apply remediations.

Another quite remarkable feature of OpenSCAP together with the XCCDF was the support for running practically any scripted action during compliance checks, which

could effectively expand the usability of OpenSCAP also to reactive defensive mechanisms.

3.5 Attack Graph Analysis Logic

Wang et al. (2008) and Zhang et al. (2011) argued that intrinsic metrics defining the severity of the vulnerabilities through CVSS are not sufficient for security measurement in contexts such as network environments which contain complicated configuration, where an overall security posture of the whole network environment needs to be established. Similar findings have been established in research on attack modelling through graphs, for instance, by Gallon & Bascou (2011), Ingols et al. (2009), Kotenko & Chechulin (2013), Lu et al. (2009), Ou et al. (2005, 2006), Wang et al. (2007, 2008, 2011).

Many of the above research included logic-based programs to model such attack graphs. Logic programming, often referred to as declarative style of programming is quite unique paradigm in that, the programmer defines only what needs to be computed without explicitly specifying how to compute it. The capability of finding solutions to such given problems is in the logic-based program's ability to logically deduce facts using sets of rules which are defined in the logical statements of the program itself (Pfenning, 2007). It is left for the interpreter to decide how to perform the computation. In contrast, in procedural style programming, languages such as C and Java, the program explicitly describes the procedures, routines and subroutines for every series of computational steps of the computation. (Smaill, 2015.)

3.5.1 Prolog

Prolog (an abbreviation for PROgrammation en LOGique) is a logic programming language invented in 1972 by Alan Colmerauer, Robert Kowalski and Philippe Russel. It was originally designed to process natural language, performing deductions based on a text written in French. The man-machine communication system was the first large Prolog program ever to be written, which quickly evolved to a theorem-proving programming language (Colmerauer & Roussel, 1992.)

Prolog combines the concepts of logical and algorithmic programming, and is recognized not just as an important tool in AI and expert systems, but also as a general purpose high-level programming language with unique features such as unification and backtracking (ISO/IEC 13211-1:1995).

Prolog language is based on a set of mechanisms such as pattern matching, tree-based data structuring and backtracking that make it well suited for symbolic, non-numeric problems involving objects and relations between them. (Bratko, 2011.)

The way Prolog programs are written and how they are interpreted, both syntactically and semantically are defined in the ISO/IEC Prolog Standard (ISO/IEC 13211:1-1995). Prolog uses syntax of First-order predicate logic, in which formulas are written in so-called *clause form* (a conjunctive normal form in which quantifiers are not explicitly written), and are further restricted to Horn clauses only that have at most one positive literal (Bratko 2011).

A Prolog program consists of one to many clauses, which in Prolog is the term for an inference rule with a `Head :- Body` structure (Pfenning, 2007). Clauses can be facts, rules or questions. A clause that only has the head and no body structure is considered a fact whereas a clause with only body and no head structure is considered a question (Bratko, 2011).

A fact is a clause that always holds true regardless of the conditions of the domain. Clauses that have the `Head :- Body` structure, rules, hold true only conditionally (Bratko, 2011).

The algorithm for Prolog's way of answering questions, satisfying goals, by Bratko (2011) is in Appendix 1.

Semantically, the clauses in the Prolog program form the base knowledge of the existing "*world*" for the logic engine. The knowledge base is basically a collection of known facts and rules against which the logic engine tries to prove queries, and also to deduce new knowledge (Bratko, 2011).

The common logical operands in the knowledge bases for Prolog programs are expressed according to the following table:

Table 2. Logical operands in Prolog (Bratko, 2011)

Logical operation	Translation	Prolog operator
implication	if	:-
conjunction	and	,
disjunction	or	;
negation	not	\+

A Prolog equivalent to the well-known modus ponens rule of inference is written in the following knowledge base, a sample program called `modusponens.P`:

```
q(X) :- p(X).        %rule which states that
                     %∀XP(X) → Q(X)

p(e).                %fact which instantiates variable
                     %X with a chosen value e of the
                     %domain
```

In Prolog prompt, after the above knowledge base was loaded, the conclusion of $q(X)$ could be queried by simply entering:

```
?- q(e).
yes
```

Above, Prolog answered "*yes*". Prolog evaluated the query and the truth of $q(e)$ and logically deduced the goal from the program.

If the query contains variables, Prolog also has to discover for which of those variables the goals can be satisfied. If none of the instantiation of the variables satisfy the goals, Prolog will simply answer to the query "*no*" (Bratko 2011).

Returning to the `modusponens.P` example, all instantiations of the variable X of our program are to be listed. This is achieved with the following query:

```
?- q(X).
```

Here Prolog found an instantiation for the variable X, and produced an answer:

```
X = e
```

This is only the first of the answers. To retrieve the rest of the possible answers a semicolon, the Prolog operand for logical disjunction "`;`" was entered after the previous answer:

```
X = e;
no
```

Now Prolog answered "*no*". Prolog found no additional elements in the program for which the goal would satisfy. This is logical, since no other instantiations of the variable X existed in the knowledge base.

One of Prolog's powerful abilities is recursion with which the program is able to derive new facts from the previous solutions within the same query when a predicate contains a goal that refers to itself (Bratko, 2011). Consider the following example program `link.P` in which there are five nodes a1…a5:

```
link(a1,a2).
link(a2,a3).
link(a3,a4).
link(a4,a5).
connected(X,Y) :- link(X,Y).
connected(X,Y):- link(X,Z),connected(Z,Y).
```

The above program consists of four facts and two rules. The four facts define that there exist a link of some sort between nodes a1 and a2; a2 and a3; a3 and a4; a4 and a5, respectively. The concept of connection is defined in the subsequent rules. The first rule states that *X* and *Y* are connected if there exist a link between *X* and *Y*. The recursion occurs in the second rule of the program, which states that *X* and *Y* are connected if there exists a link between *X* and *Z*, and *Z* and *Y* are connected. The connection is defined with a recursion, with a goal referring to itself. Using the two rules the program is able to deduce all connections for the nodes in our example. Below, Figure 5 is a logical depiction of the `link.P` program, where the direct and recursed connections are shown as arrows:

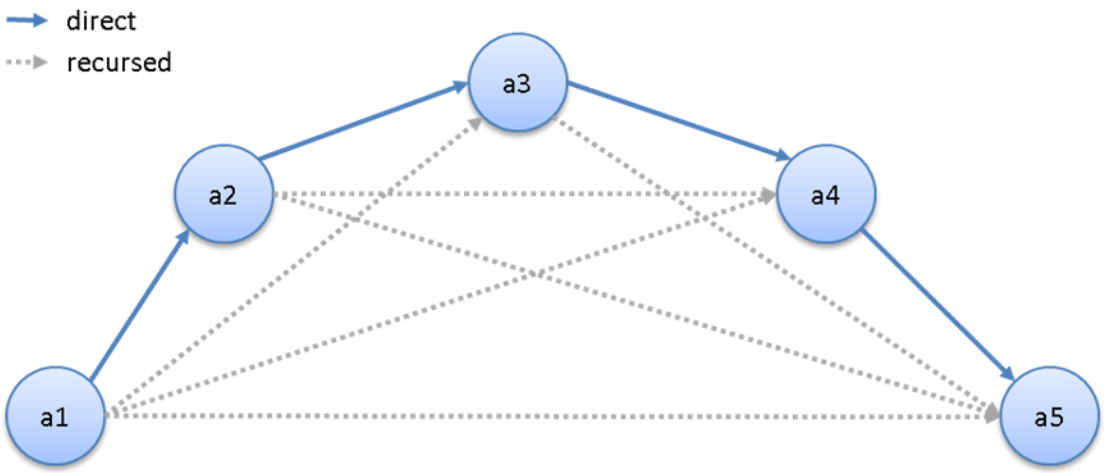

Figure 5. Direct and Recursed Connections of link.P program

Logically the order in which the proofs are searched should not be relevant (Pfenning, 2007). Both Pfenning and Bratko (2011) find, however that the order by which the goals and clauses are presented in the knowledge base, greatly affects the efficiency of the Prolog query resolution. This is demonstrated by following Bratko's (2011) examples with two additional variations of the `link.P` example. In comparison with the original program, *variation1* reverses the order of the clauses and *variation2* reverses the goals of the second clause:

variation1:
```
connected(X,Y) :- link(X,Z),connected(Z,Y).
connected(X,Y) :- link(X,Y).
```

variation2:
```
connected(X,Y) :- link(X,Y).
connected(X,Y) :- connected(X,Z),link(Z,Y).
```

By instructing Prolog to enable tracing, every step of the program execution is shown. Below is the program trace for the question "*does a2 have a connection to a4*", translated to Prolog input as a query: *connected(a2,a4).*

```
?- trace.
yes
[trace]

?- connected(a2,a4).
(0) Call: connected(a2,a4) ?
(1) Call: link(a2,a4) ?
(1) Fail: link(a2,a4) ?
(2) Call: link(a2,_h236) ?
(2) Exit: link(a2,a3) ?
(3) Call: connected(a3,a4) ?
(4) Call: link(a3,a4) ?
(4) Exit: link(a3,a4) ?
(3) Exit: connected(a3,a4) ?
(0) Exit: connected(a2,a4) ?
yes
```

In the trace above, as Bratko (2011) explained, Prolog first *(0)* executed the initial query as its primary goal `connected(a2,a4)`. Following the first rule of the knowledge base, it continued to *(1)* execute the subgoal `link(a2,a4)`. The

knowledge base did not contain fact for such direct link, so the subgoal failed. Prolog then followed the second rule of the knowledge base and *(2)* executed the goal `link(a2,_h236).` Here the `_h236` is a temporary substitute for the variable Z in the second rule of our program. It is a free variable that is used only once in a clause, and can be instantiated with any value. From the trace we can see that the goal succeeded with the free variable now having `a3` (the only direct link to `a2`) as its instantiation. Next, the program continued to *(3)* execute the second goal in the body of the second rule of our program, `connected(a3,a4).` Again following the first rule of the knowledge base, Prolog continued to *(4)* execute the subgoal `link(a3,a4)`. This time the knowledge base contained a fact for such link, so the subgoal, and also the second goal (3) succeeded. Finally the primary goal *(0)* also succeeded, and Prolog answered to the query "*does a2 have connection to a4*" with:

yes.

Interestingly, even if all three variations of the same program are semantically equal to one another, the goal satisfaction with *variation1* is inefficient compared to the other two, which perform equally efficiently. This is because reversing the order of the clauses, as in *variation1*, makes Prolog search for the solution first by looking for such additional nodes that exist between the two, and also have links to both of them, rather than trying to establish whether the queried connection arguments immediately satisfy the corresponding link (Bratko, 2011.)

Bratko also underlines that it is important to realize that it is possible to cause Prolog programs to run indefinitely. This may occur in situations which Prolog tries to find the answer by choosing wrong path in the process, leading to an infinite loop the program is unable to escape from (Bratko 2011). Following Bratko's examples, such situation was established with yet another variation of the `link.P` program. If the order of the goals and also the order of the clauses were reversed to the original version, Prolog would end up running the program until it ran out of memory, not able to deduce the answer:

variation3:
```
connected(X,Y) :- connected(X,Z),link(Z,Y).
connected(X,Y) :- link(X,Y).
```

The declarative meaning of the program was unchanged and was shared among all versions: there is a connection from *X* to *Y* if there is a link from *X* to some node *Z*, and there is a link from *Z* to *Y*, or if there is a link from *X* to *Y*. Regardless of the declaratively correct definition of *variation3*, Prolog was unable to answer the same query: `connected(a2,a4).` (Bratko, 2011.)

3.5.2 Datalog

Datalog is a declarative logic programming language in which each formula is a function-free Horn clause (Datalog User Manual). Datalog does not allow function symbols as arguments, meaning that the so-called Herbrand universe of ground instances of predicates is infinite (Pfenning, 2007). Datalog terms must be variables or be drawn from a fixed set of constant symbols. Another restriction is that any variables used in the head of a clause also have to exist in a nonarithmetic positive in the body of the rule (Pfenning, 2007).

An advantage in Datalog syntax over Prolog is that it does not limit the order of the clauses, and the queries are quaranteed to terminate. It uses an efficient evaluation method by binding the start and goal stop and deducing every possible answer in between (Datalog User Manual, 2002).

3.5.3 XSB

XSB is a module-centric research-oriented logic programming and deductive database system with semantically enriched superset of Prolog. XSB is compatible with both ISO-Prolog and Datalog programs. It includes enhancements which allow for tabling with negation and higher-order logic programming (XSB Documentation).

XSB has two ways of evaluating predicates. Prolog-style evaluation, and tabling or tabled resolution. The ability of tabled resolution provides more declarative programs than Prolog. Additionally, the ability to store calls to tabled predicates in a searchable structure with their proven instances, and to compile predicates as tabled, allows for the programs to properly terminate with an answer (XSB Documentation).

The concept of tabled resolution is that it governs the procedure calls in a way that it remembers every call and also the answers that are returned. If a similar call is made again, the previously deduced answers are used to satisfy the new request. That way the same procedure call is not performed twice (XSB, Documentation).

Recall the program that caused Prolog to run in a loop indefinitely, *variation3* of the program `link.P` from chapter 3.4.1. When the same program is run with XSB and when the interpreter is instructed to use tabling, XSB ensures that all calls to the predicate `connected` are tabled throughout the program (XSB Documentation):

```
:- table connected/2.
connected(X,Y) :- connected(X,Z),link(Z,Y).
connected(X,Y) :- link(X,Y).
```

Given the complete program above, XSB was able to deduce the correct answer to the same query that effectively caused Prolog to enter infinite loop as shown in the following:

```
?- connected(a2,a4).
yes
```

The ability to solve recursive queries has proven useful in number of areas, such as deductive databases, language processing, program analysis, model checking and diagnosis (XSB, Documentation).

XSB's support for higher-order logic programming allows for programs that have complex terms as predicates. Higher-order enhancements in XSB enable incorporation of some higher-order constructs in a declarative way within logic programs, while retaining first-order declarative semantics (XSB, Documentation).

3.6 Attack Graph Analysis Engine

The main utility for the data analysis component used in this research was a logic-based data-driven network security analyzer, MulVAL. MulVAL research tool stands for Multihost, Multistage Vulnerability Analysis Language. It is free software, released under GNU GPL version 3, and is developed and maintained by Kansas State University (MulVAL, Argus CyberSecurity Lab, Kansas University).

MulVAL was chosen as the main utility largely because of its past presence in several attack graph-related research projects. For instance, MulVAL has appeared in research on scalable approach to attack graph generation (Ou et al. 2006), on logic-based network security analyzer (Ou et al. 2005), on measuring overall security of network configurations (Wang et al. 2007) and on measuring security risks on networks using attack graphs (Wang et al. 2010).

In addition to the existing research, MulVAL seemed particularly suitable for this research, too, due to its versatile configurability in custom network environments.

3.6.1 MulVAL Framework

MulVAL uses Datalog as its modelling language for the elements in the analysis. The main idea behind MulVAL was that most configuration information can be presented as Datalog facts, and most attack techniques and OS security semantics can be specified using Datalog rules (Ou et al. 2006). The logic inside MulVAL uses XSB to evaluate the Datalog interaction rules against the input data. XSB environment was chosen for the analysis in MulVAL because it supported tabled resolution of facts (Ou et al. 2005).

Architecturally the framework for the version of MulVAL used in this Thesis is described by Ou et al. (2005), and is depicted in Figure 6.

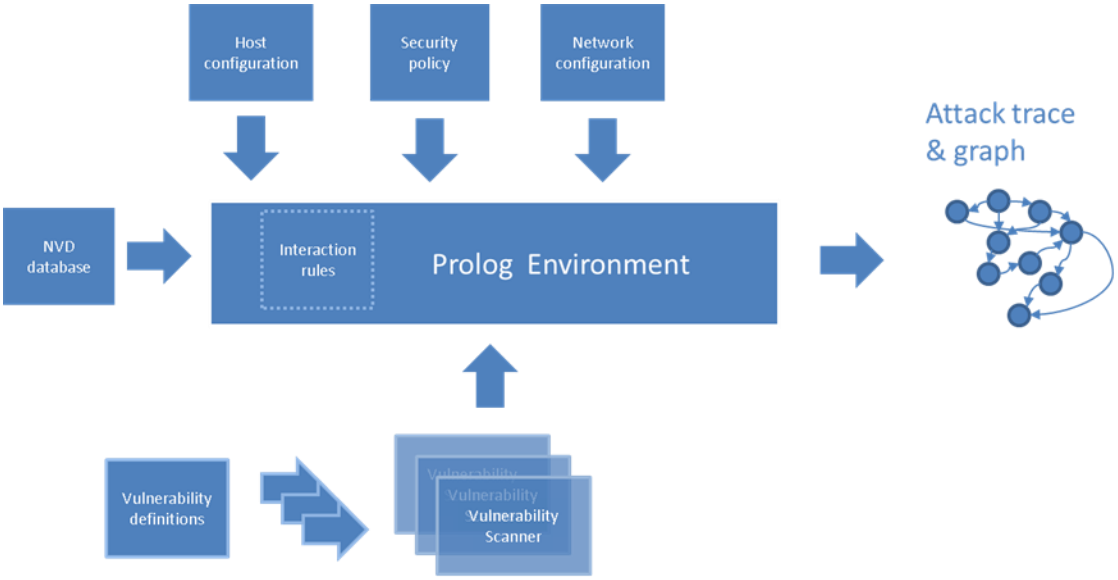

Figure 6. MulVAL Framework (Ou et al. 2005)

MulVAL logically evaluates and combines all the various elements: the network configuration, system-specific software and services configuration and the user rights policies, and then iteratively applies interaction rules on the combined input data. The Interaction rules define how the analysis will model the effect of the discovered vulnerabilities. Inside the interaction rule set, predicates are declared in both Datalog-style and as tabled predicates. The Datalog-style predicates are read from the translated vulnerability assessment result file, and the tabled predicates are used in the recursive deductions, characteristic to XSB (MulVAL Readme). The interaction rules can be customized to accommodate different network environments.

3.6.2 MulVAL Input Data Types

MulVAL consumes several individual sources for information on the state and configuration of the environment, the elements, all of which are encoded as Datalog facts. There are four main types of information that form the input data for MulVAL: 1) the vulnerabilities that are known to exist and are reported; 2) the vulnerabilities that exist the hosts and devices; 3) the outstanding configurations for the software and services running on all the hosts in the environment and; 4) the configuration of the network, the access-lists for the hosts and their services within the environment. Additional information types include user credentials and their use policy in the

environment and the interaction rules, which define how all the input types interact (Ou et al. 2005).

For the analysis, the first three input data types are mandatory. MulVAL will require a list, a database, of the all the vulnerabilities that exists in the domain of discourse that could also exist inside the environment to be analyzed. For this, MulVAL uses its own, internal database into which the vulnerability information will be pulled from NVD manually using the *nvd_sync.sh* script. The internal database is used in conjunction with the vulnerability assessment result file when the engine will begin deducing the existence of possible vulnerabilities and the paths for their potential exploitations in between the hosts in the environment.

Vulnerability scanners such that support credentialed authentication during the assessment scans are able to produce result files that contain the elements for the input types 2 and 3. The quality and precision of the software and service level results for input data type 3 is dependent on the accuracy of the policy, compliance or best practices template of the vulnerability scanner, based on which the result will be generated. MulVAL supports input files of type 2 and 3 in XML format. More specifically, Nessus and OVAL XML file types are supported (MulVAL Readme).

The analysis can be performed with the just the first three input file types, however, the ability to also use additional information types will enable much more precise analysis output. For instance, without specifying input data type 4, network configuration or, access-lists, MulVAL will assume that each host or device is able to connect with every other host or device in the environment. By providing accurate host-to-service access-lists, MulVAL will produce much more usable analysis results.

The two additional input data types include the users and their access policies in the network environment and, interaction rules (Ou et al. 2005).

An example of an interaction rule for a remote exploit of a privilege-escalation vulnerability in a service program, represented as a Datalog rule, by Ou et al. (2006):

```
execCode(Attacker, Host, User) :-
networkService(Hpst, Program, Protocol, Port, User),
```

```
vulExists(Host, VulID, Program, remoteExploit,
privEscalation),
netAccess(Attacker, Host, Protocol, Port).
```

MulVAL interaction rules, such as above, are written in Prolog form, where the first line represents the goal and the following lines represent facts that will satisfy the goal. Capitalized letters represent free variables, which can be instantiated with any term. The interaction rule has the following meaning: if a Program, running as User on Host as a service, listening on Protocol and Port, contains a vulnerability that is remotely exploitable, and whose impact is privilege escalation, and the attacker can access the service through network the network, then he can execute arbitrary code on the Host as User (Ou et al. 2006.)

In MulVAL, predicates for input types 2 and 3, are *primitive*, and they represent configuration information reported by vulnerability scanners. Predicates such as execCode and netAccess are *derived* and they are computed from the configuration information (Ou et al. 2006).

3.6.3 MulVAL Analysis and Graph Building Algorithm

The analysis algorithm in MulVAL is divided in two phases: 1) Attack simulation and 2) policy checking. In the attack simulation phase, all possible data accesses that can result from multistage, multihost attacks are being derive through the Datalog program (Ou et al. 2005):

```
access(P, Access, Data) :-
dataStore(Data, H, Path),
accessFile(P, H, Access, Path).
```

The meaning above is that, if `Data` is stored on machine H under `Path`, and user P can access files under the `Path`, then P can access the `Data`. The attack simulation occurs in the derivation of `accessFile`, which involves the Datalog interaction rules and data inputs from various components of MulVAL. In the policy checking phase,

the data access tuples output from the attack simulation phase are compared with the given security policy. If access is not allowed by the policy, a violation is reported. The following Prolog program is in charge of the policy checking (Ou et al. 2005):

```
policyViolation(P, Access, Data) :-
access(P, Access, Data),
not allow(P, Access, Data).
```

For abstractions of the attack paths, MulVAL uses an analysis algorithm, such that returns all possible attack paths. To achieve the computational goal, the analysis engine must traverse all possible derivation paths. While performing the derivations, MulVAL also records every step in the process by utilizing tabled execution. (Ou et al. 2006).

For the traversal of all of the derivation paths, another sub-goal is implemented. The additional sub-goal will call the assert_trace() function, which, during a successful evaluation of a rule, records all successful derivations into a temporary trace file, eventually allowing for the logical attack graph to be constructed (Ou et.al, 2006).

The definition for an attack simulation trace has the following format (Ou et al. 2006):

```
TraceStep ::= because(interactionRule, Fact,
          Conjunct)
Fact      ::= predicate(list of constant)
Conjunct  ::= [list of Fact]
```

With the addition of the new subgoal, the interaction rule from 3.5.2. now had the following presentation (Ou et al. 2006):

```
execCode(Attacker, Host, User) :-
networkService(Hpst, Program, Protocol, Port, User),
vulExists(Host, VulID, Program, remoteExploit,
privEscalation),
netAccess(Attacker, Host, Protocol, Port),
```

```
assert_trace(because('remote exploit of a server
program', execCode  (Attacker, Host, User),

[networkService(Host, Program, Protocol, Port, User),

vulExists(Host, VulID, Program, remoteExploit,
privEscalation),

netAccess(Attacker, Host, Protocol, Port)])).
```

Finally, MulVAL's graph building algorithm is depicted in Figure 7, in which every *TraceStep* term becomes a derivation node in the attack graph. The *Fact* field in the trace step becomes the node's parent and the *Conjunct* field becomes its children (Ou et al. 2006):

Input: set τ containing all the *TraceStep* terms,
attacker's goal G
Output: logical attack graph (N_r, N_p, N_d, E, L, G).

1. $N_r, N_p, N_d, E, L \leftarrow 0$
2. For each $t \in \tau$ {
   let $t = because(interactionRule, Fact, Conjunct)$
3. Create a derivation node r
   $N_r \leftarrow N_r \cup \{r\}$
   $L \leftarrow L \cup \{r \rightarrow interactionRule\}$
4. Look up $n \in N_d$ such that $L(n) = Fact$,
5. If such n does not exist
   {
   create a new fact node n
   $L \leftarrow L \cup \{n \rightarrow Fact\}$
   $N_d \leftarrow N_d \cup \{n\}$
   }
6. $E \leftarrow E \cup \{(n, r)\}$
7. For each fact f in $Conjunct$ {
8. Look up fact node $c \in (N_p \cup N_d)$ such that
   $L(c) = f$,
9. If such c does not exist
   {
   create a new fact node c
   $L \leftarrow L \cup \{c \rightarrow f\}$
   If f is primitive { $N_p \leftarrow N_p \cup \{c\}$}
   else { $N_d \leftarrow N_d \cup \{c\}$}
   }
10. $E \leftarrow E \cup \{(r, c)\}$
    }
    }

Figure 7. Attack Graph Building Algorithm (Ou et al. 2006)

3.6.4 Attack Graph Construction

The constructed logical attack graph depicts the combinations of different vulnerabilities and how they may be put together to conduct a multistage attack in an environment in which the assets have dependencies. Figure 8 shows the MulVAL architecture of the logical attack graph construction (Ou et al. 2006).

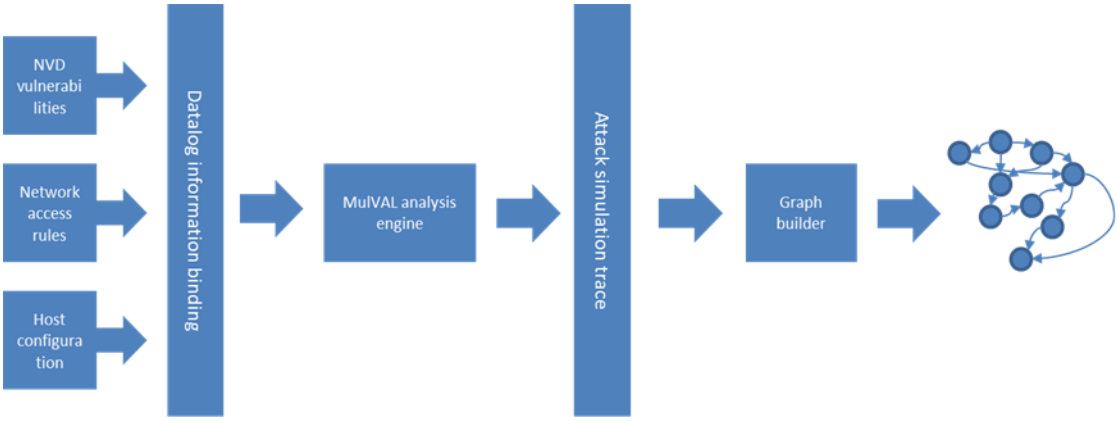

Figure 8. Architecture for the Logical Attack Graph Generator (Ou et al. 2006)

MulVAL can be used to construct an attack graph in two ways. Either 1) running the attack-graph generator directly by invoking the graph generation script with an input file or; 2) by using adapters to prepare an input file for the attack-graph generator (MulVAL Readme).

In the latter case, an input file will be created after performing a translation on the vulnerability assessment result file. The built-in *nessus_translate* script, for instance, will walk the Nessus XML result file and look for vulnerabilities that are listed. The script goes through the vulnerability details as to how they are exploitable, their CVSS scoring and access complexity, and their existence in the assessed environment. Finally, the script will list every discovered services running on the hosts in the environment, and how they are reachable from outside and also laterally, inside the environment. This information is then translated into MulVAL as Datalog clauses, as components to the knowledge base.

By default, the attack graph will be generated in .TXT and .XML formats. By instructing the graph generator to perform visualization, the graph will be saved also in .CSV, .EPS and .PDF formats.

3.6.5 Grouping Algorithms

Zhang et al. (2011) find it is quite possible even in relatively small environments that the attack graph will become large, contested and visually too dense to interpret. According to them, various techniques and approaches have been introduced in past research to both, addressing the visualization challenge, and using traditional method to produce an attack graph without taking targets' similarities into account (Zhang et al. 2011).

MulVAL utilizes special algorithms by Zhang et al. designed for creating abstract network models for large-scale networks based on network reachability and host configuration information. In their work they find that the abstracted models dramatically reduce the complexity of the attack graphs by improving the visualization but also correcting a possibly distorted quantitative vulnerability assessment result (Zhang et al. 2011).

A caveat presented by Zhang et al. underline that their proposed abstractions are suitable for risk assessment on a macroscopic level of an enterprise network. They find that abstraction inevitably loses information which in some cases may lead to not catching such subtle security breaches that may occur due to, for instance, misconfiguration of a single host which is falsely considered similar to a group of hosts (Zhang et al. 2011).

The network abstraction models are applied in three steps: 1) reachability-based grouping, 2) vulnerability grouping and 3) configuration-based grouping.

Reachability-based grouping

In reachability-based grouping, all hosts are being grouped based on their reachability information, host access list (hacl). All hosts in the same reachability group can be abstracted as a single node, and the grouping is applied to all assets in

the same subnet. Algorithm for reachability-based grouping is in Appendix 3 (Zhang et al. 2011).

Vulnerability grouping

In MulVAL, vulnerability grouping is conducted based on the application-level identification. Zhan et al. (2011) established that a single host can contain dozens or even hundreds of vulnerabilities, all of which may appear in a distinct attack graph to further compromise the system. They claim that not all those paths provide unique information since many of such vulnerabilities are similar in nature. Zhang et al. (2011) also claim that it is more important, at a higher level to know that some vulnerability in an application could result in a security breach, rather than enumerating all the distinct but similar attack paths, since vulnerabilities in the same application are often exploited by the same mechanisms.

In MulVAL, vulnerability grouping will display the vulnerable applications instead of a list of CVE-numbers. The vulnerability grouping algorithm assigns the highest metric to indicate aggregated vulnerability score after the grouping, with the ability to alter the aggregation method through customization (Zhang et al. 2011). The algorithm for vulnerability grouping is in Appendix 4.

Configuration-based grouping

Configuration-based grouping in MulVAL will iterate over all hosts in the same reachability group and record their configuration information. If the discovered configuration matches one previously recorded, the new information will not be recorded in the set. The result set of the algorithm will only contain unique representative hosts for each group of hosts in the same reachability and configuration (Zhang et al. 2011). The algorithm for configuration-based grouping is in Appendix 5.

3.6.6 A Practical Example

One host in the test network was taken as an example. This server runs as the platform for the Information Services Environment (ISE), the internal information portal. The platform is a modular object-based publishing environment on top of a Windows Server 2003 operating system. Nessus vulnerability scanner detected and identified a known vulnerability in one of the services running on the server. After running the translation script, the vulnerability information was presented to MulVAL with the following clause structure:

```
vulExists(ISEserver,’CVE-2003-
0352’,windows_2003_server).

vulProperty('CVE-2003-
0352',remoteExploit,privEscalation).

cvss(‘CVE-2003-0352’,1).

networkServiceInfo('ISEserver',windows_2003_server,tc
p,'445',someUser).

hacl(_,_,_,_).
```

Looking at the translations more closely;

```
vulExists(ISEserver,’CVE-2003-

0352’,windows_2003_server).
```

Here, the existence of a vulnerability that was discovered during vulnerability assessment is presented as a Datalog fact. This information does not contain detailed information on how the vulnerability can be exploited, only that it exists in the host. Depending on the method of the utility that performed the assessment, the existence may be have been decided based on the version of the discovered service or, it may also have been confirmed through functional testing of the service.

The property information of the discovered vulnerability offer more information on the vulnerability's exploitability with regard to access vector and impact type. From CVSS v2 severity scoring, also the metric for access complexity is recorded, and the respective information is then converted as a Datalog fact into MulVAL:

```
vulProperty('CVE-2003-
0352',remoteExploit,privEscalation).

cvss(‘CVE-2003-0352’,l).
```

The first of the two new Datalog facts defines that the discovered vulnerability is prone to being exploited remotely which can then result in privilege escalation. The second new fact define that the required access complexity is low.

Next, the input information of the software and services running in the host are translated as Datalog facts. Continuing with the ISE example:

```
networkServiceInfo(ISEserver,windows_2003_server,tcp,
445,someUser).
```

The clause structure above defines that a system service is running on host *ISEserver*, uses TCP protocol, listens on port 445 and is being run with *someUser* privileges.

Network configuration of the environment, containing router, firewalls, switches and their respective broadcast domains are modeled as host access lists HACL. Unless specified in the input file, MulVAL assumes each host has connection to every other host in the environment. The corresponding Datalog fact for such access list is:

```
hacl(_,_,_,_).
```

Above, the free variable is used to indicate full connectivity for MulVAL. Should the example include more complex evaluation, the free variables would enable the recursive use of the same access rule across the whole program.

4 Construction

4.1 Thesis Test Network

The test network in this thesis is a semi-isolated environment (Figure 9.) in which there are four logical network segments *auth, core, ise* and *monitor*. Semi-isolated in this case means that logically there is no connection from the outside of the perimeter firewall to any of the respective network segments. This is, however, not

an explicit condition. The test network environment is constructed in a way that it mimics a mission network environment, such that are deployed into different operations and such that enables for candidate C2 systems to connect to and consume services that are being produced.

Candidate C2 systems apply for an Information Assurance (IA) assessment before they are able to connect. Due to the connection window during the candidate C2 systems' IA process, there is a possibility that the network environment will become exposed to external networks by proxy, and is therefore considered only semi-isolated.

Additionally, a separated management network (*mgmt*) is deployed for all assets in the environment to enable maintenance tasks on the assets. The management network is separated, in that every asset is equipped with another network interface dedicated to management use. The Kernel-based Virtual Machine (KVM) hosts and guests do not, however, use kernel isolation or kernel separation in this construction.

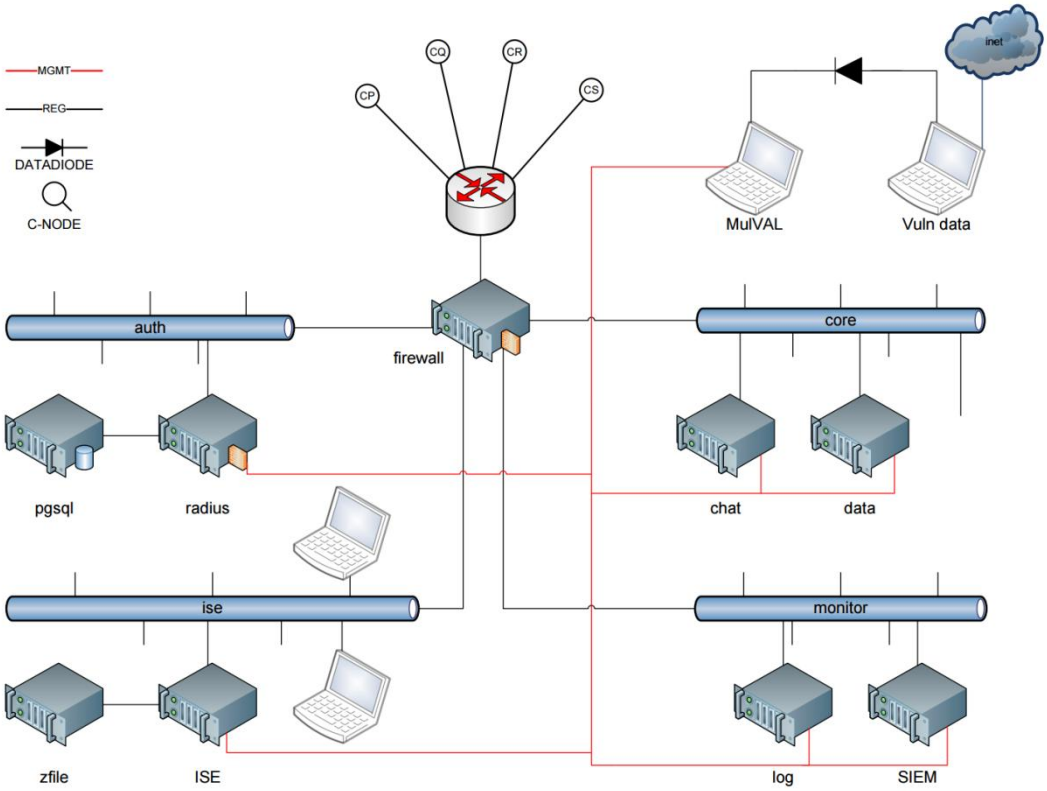

Figure 9. Logical depiction of the thesis test network (HMN)

4.2 Access-lists

Table 3 shows the hosts and their respective firewall rules, required for the availability of the HMN service set in the four network segments:

Table 3. Thesis Network Host Access-lists

HOST	Network segment	Allowed ports & protocols
RADIUS	AUTH	1812/UDP, 1813/UDP
CHAT	CORE	5222/TCP, 5223/TCP, 6667/TCP, 777/TCP, 9090/TCP, 22/TCP
DATA	CORE	80/TCP, 443/TCP, 22/TCP
ISE	ISE	80/TCP, 8085/TCP, 137/TCP, 137/UDP, 138/UDP, 139/TCP, 445/TCP
LOG	MONITOR	514/UDP, 514/TCP, 10514/TCP
SIEM	MONITOR	80/TCP, 443/TCP, 22/TCP, 514/UDP, 514/TCP, 1514/UDP, 162/UDP, 12000/UDP, 40001-40005/TCP*, 40011/TCP*, 555/UDP*, 6380/TCP*, 9390/TCP*, 33800/TCP*

* the ports marked with an asterisk are required if the SIEM sensor is being deployed as a separate unit instead running all the components in one SIEM installation (Alienvault® USM™ Depoyment Guide).

4.3 Vulnerability Information

4.3.1 MulVAL statistics

The vulnerability related information utilized in the MulVAL analysis engine used the feed from NVD as the baseline. More specifically, the NVD XML feed with version 1.2.1 schema.

Prior to synchronizing the NVD vulnerability feed, the statistics in NVD page (NVD Home) showed:

- 77 607 CVE vulnerabilities
- 356 Checklists
- 249 US-CERT Alerts
- 4 433 US-CERT Vuln Notes
- 10 286 OVAL Queries
- 113 937 CPE Names

The NVD repository and their CVSS and CPE analysis amendments may well be the most referenced collection of MITRE's CVE data, but there has been some scrutiny about the completeness of the CVE and NVD, at least from a commercial competitor (Risk Based Security, 2015). While being a commercial ad, really, the article questions the coverage of MITRE's CVE dictionary. Due to lack of funding resources for the thesis, it was not possible to look into the coverage of the Risk Based Security's *VulnDB*, or Rapid7's vulnerability database or any other commercial vulnerability databases to find out whether the claim had any grounds.

It was possible, however, to take a closer look into the contents of the NVD repository. The statistics from both NVD and *cvedetails.com* were put for comparison in the Excel chart, in Figure 10.

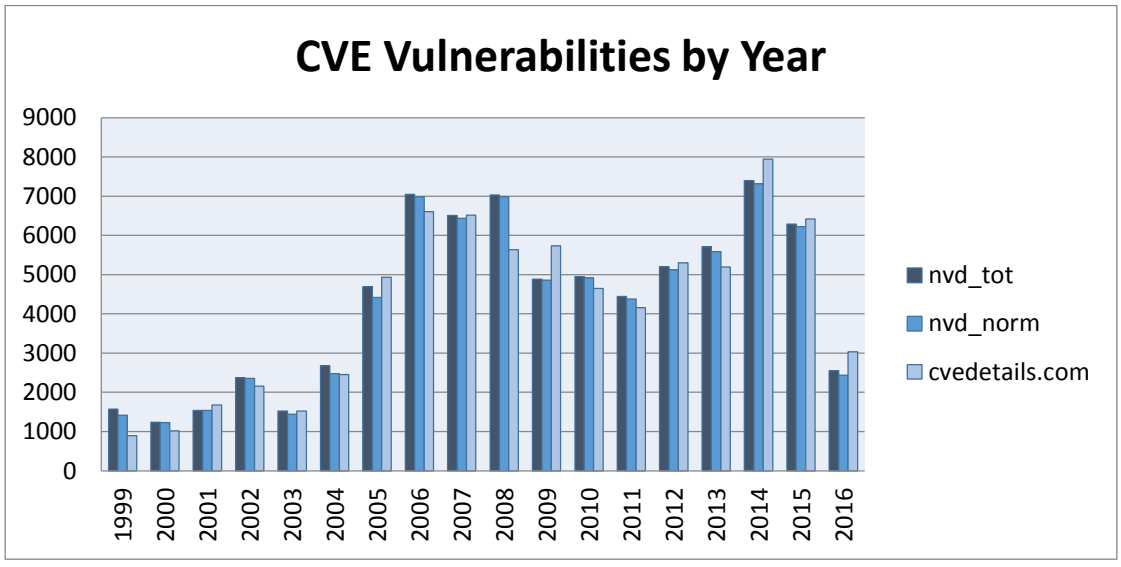

Figure 10. CVE Vulnerabilities by Year

After performing the synchronization script *nvd_sync.sh* , the total number of vulnerabilities reported in the MulVAL's internal database, *nvd_tot*, equaled to the number of CVE vulnerabilities in the NVD website (NVD Home), as seen from the following database query:

```
select count(*) from nvd;
+----------+
| count(*) |
+----------+
|    77607 |
+----------+
1 row in set (0.03 sec)
```

Inside the NVD data, there were some inconsistencies with the CVE records. Some of the records were obviously meant for testing purposes, for instance regarding CVE ID syntax change (CVE - CVE ID Syntax Change, 13.9.2016). Some were duplicates, or records that have wrong id number, or contained incomplete confidentiality, integrity or availability impact categorization information, and so forth.

```
<entry type="CVE" name="CVE-2014-59156" seq="2014-59156"
published="2015-01-13" modified="2015-01-13" reject="1">
    <desc>
      <descript source="cve">** REJECT **  DO NOT USE THIS
CANDIDATE NUMBER. ConsultIDs: CVE-2014-2352.  Reason: This
candidate is a duplicate of CVE-2014-2352.  The wrong ID was
used.  Notes: All CVE users should reference CVE-2014-2352
 instead of this candidate.  All references and descriptions in
this candidate have been removed to prevent accidental
usage.</descript>
    </desc>
```

Incomplete or redundant or duplicate records, such as the one above, distort the statistics of real vulnerabilities and ought to be left out of the calculations. Therefore, in Figure 10, also a normalized total number of vulnerabilities *nvd_norm* was presented. Normalization in this case means filtering out the above mentioned, undesired vulnerability records.

According to the graph the long-term trend for newly discovered vulnerabilities has been on an incline ever since the CVE vulnerabilities have started recording, from 1999. The most number of vulnerabilities in one year have been recorded in 2014. Interestingly, as noted also by Yung-Yu et al. (2011), there is a decline of three-year period from 2008 to 2011 where the frequency of newly reported vulnerabilities was indeed decreasing before again starting to increase from 2012 onwards.

The outstanding peak for the year 2014 was found to be caused by well over a thousand (1 395) Android OS and Android library based locally exploitable vulnerabilities regarding SSL server x.509 certificate verification inability.

4.3.2 OpenVAS statistics

The GSA component of the OpenVAS architecture displays the statistics of the vulnerability feed sources inside the OpenVAS. An example is shown in Figure 11 in which four dashboard elements show CVE's by CVSS severity, CVE's vulnerabilities by year, OVAL definitions by CVSS severity and OVAL definitions by class, respectively. The statistics in Figure 11 are from April, 2016.

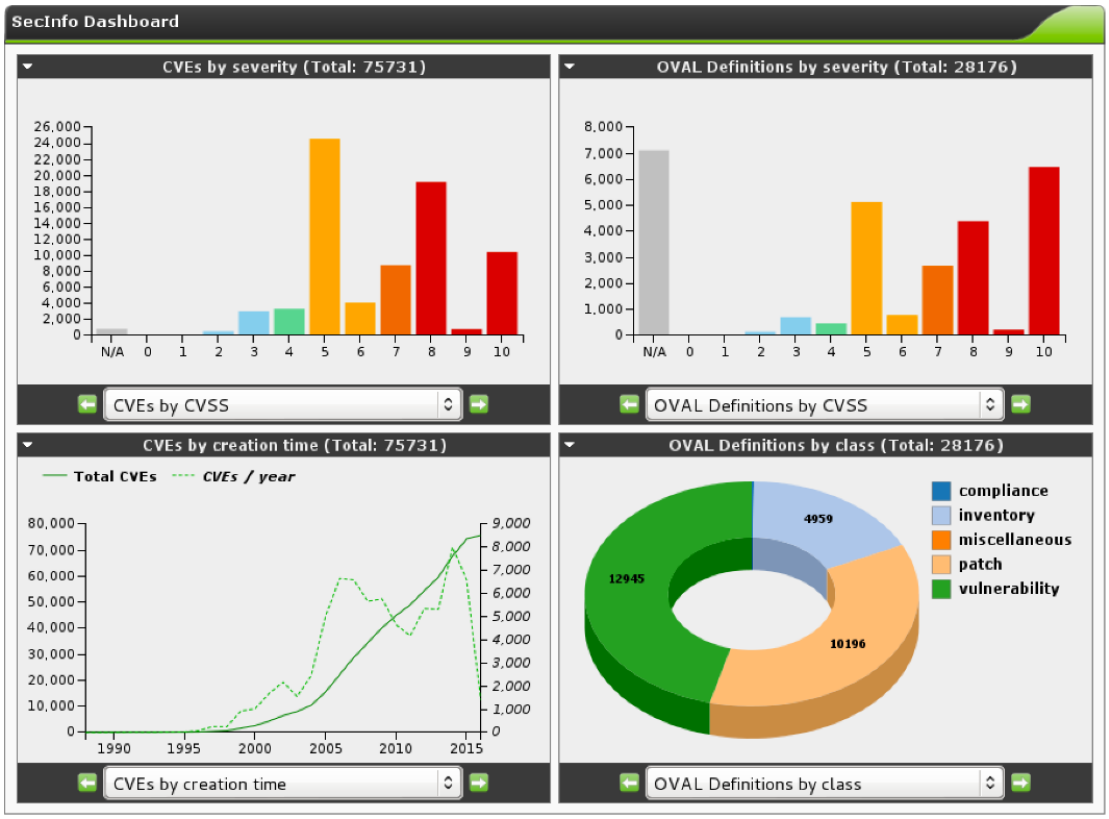

Figure 11. Screenshot of OpenVAS GSA SecInfo Dashboard

4.4 Asset Vulnerability Assessment

The main utility that was used for the majority of the vulnerability assessments in the network environment was the Nessus vulnerability scanner. The Nessus Home Feed version had a limitation of being able to scan only 16 individual IP addresses, which did not affect the scanning of the test network's assets due to the small number of them.

Where applicable, vulnerability assessments were conducted also with Nexpose, OpenVAS and OpenSCAP. While the other utilities performed vulnerability assessments, OpenSCAP was instead used to perform compliance checks on the Linux-based assets against Red Hat Enterprise Linux 6 Security Technical Implementation Guide (STIG) and customized SCAP Security Guide (SSG) checklists to look for sub-optimal configuration in the operating system or software components.

The SIEM system had the capability of running vulnerability scans with the integrated OpenVAS vulnerability scanner. This enabled for the presentation of the environment

status with regard to vulnerabilities and risk levels and also, to establish an asset database for the environment.

The results of the vulnerability assessment (depicted in Figure 12.) show that the Windows-based ISE server contained the highest number of known vulnerabilities. In total, Nessus found 177 vulnerabilities with severity ratings of low, medium, high or critical and 109 additional conditions that it flagged as informational. In comparison, OpenVAS found 387 vulnerabilities with severity ratings of medium, high or critical in the same ISE server, along with 30 informational conditions. Nexpose, the scanner that was used only with the ISE server, found 142 vulnerabilities that it flagged as medium, high or critical. Most of the critical and high level vulnerabilities were discovered in the underlying operating system suggesting poor security and patch management.

Statistically the second highest total amount of vulnerabilities was discovered in the LOG server that was used for centralized logging. Nessus found 233 vulnerabilities with severity rating of low, medium, high or critical and 67 informational conditions. OpenVAS was able to find 205 vulnerabilities and 45 informational conditions in the LOG server.

The third highest amount of vulnerabilities and almost similar results and severity profile was discovered in the RADIUS server, with vulnerabilities and informational results for Nessus and OpenVAS, 229 – 65 and 196 – 226, respectively.

Based on the vulnerability assessments with Nessus and OpenVAS, CHAT and DATA servers were the least vulnerable at least with regard to known vulnerabilities. In CHAT server, Nessus found 7 vulnerabilities of which 5 had medium severity and 2 had low severity rating. The amount of informational conditions was 61. OpenVAS found 8 vulnerabilities of which 3 were high, 5 were medium, and additional 45 for info. Similarly, in DATA server, the numbers were 8 in total of which 6 medium and 2 low and 43 info for Nessus, and 5 in total of which 2 high and 3 medium, and 31 info, for OpenVAS.

The vulnerability assessment in this thesis did not seek to compare the performance or the accuracy of the scanners used. The results from Nessus vulnerability assessments were the only ones that were further utilized in the MulVAL analysis

engine and Nexpose and OpenVAS contributed in providing enriched information for the NOC.

In addition to the vulnerability assessment, SCAP compliance evaluations were conducted on the linux-based hosts in search for sub-optimal configurations that could affect the security posture of the environment by exposing the hosts as potential stepping stones for the malefactors in their campaigns. Two different profiles were used for the SCAP compliance evaluations: SSG RHEL6 XCCDF and RedHat_6_V1R12_STIG_SCAP XCCDF.

The SSG and the STIG checklists consisted of 175 and 178 evaluation rules, respectively. The results of the evaluations were generated in both machine readable XML format and a regular HTML report. The overall scores are shown in Figure 12 as percentage bars. The bars show the relative amount of evaluated rules such that matched the ones on the implementation guidelines.

The host with the highest SCAP evaluation score was the LOG server with relative equivalence scores of 71.30 % and 50.86 % for the SSG and STIG checklists, respectively. Interestingly the same server contained the highest amount of vulnerabilities among the Linux-based hosts.

While the differences in vulnerability assessments were clear, the configuration check indicated almost similar configurations in the underlying operating system and the software base. Depicted in Figure 12, the variations for the four hosts are within 46.86 .. 50.86 for the STIG and 65.90 .. 71.30 for the SSG evaluations.

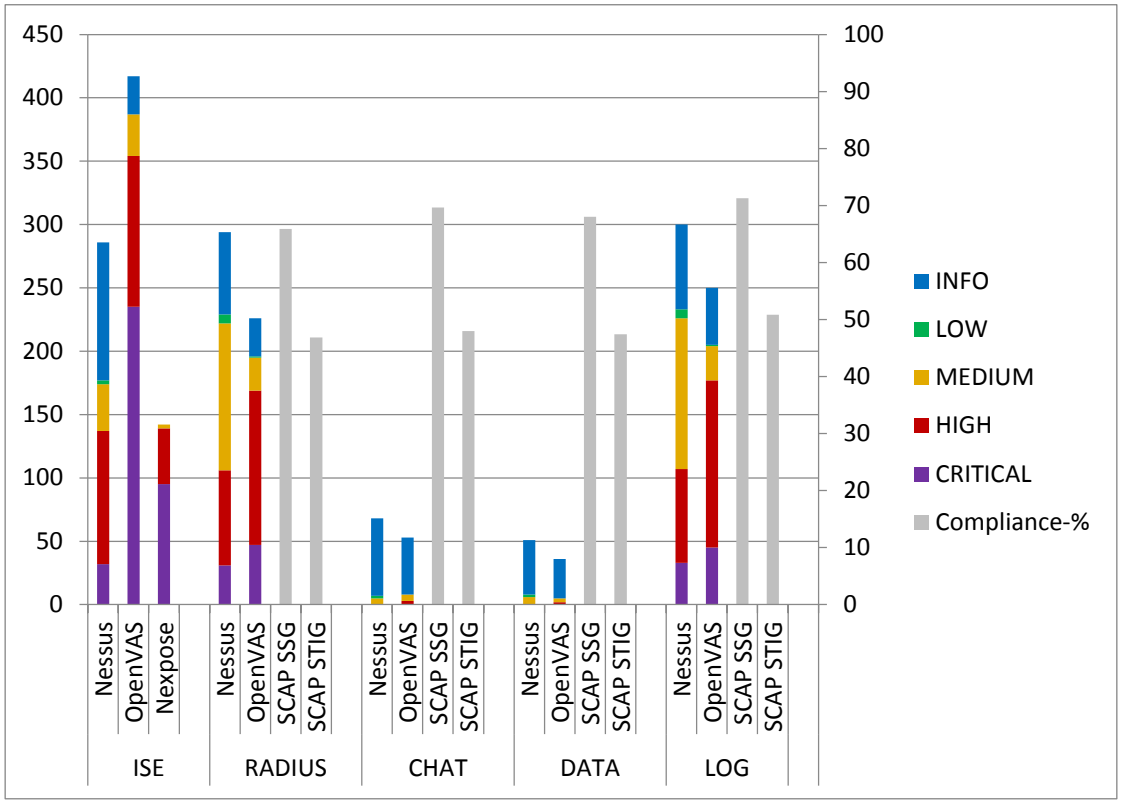

Figure 12. Vulnerability assessment of key assets

4.5 Attack Graph Compilation

The Nessus vulnerability assessment results were provided as the input data for MulVAL to process and to compile the initial attack graph for the environment. The access-lists were configured in the MulVAL configuration as they were listed in 4.2.

Due to the very high amount of vulnerabilities in three of the hosts, and the all-connecting management network, the attack graph became extremely large. Figure 13 shows just a small portion of the attack graph for the ISE server.

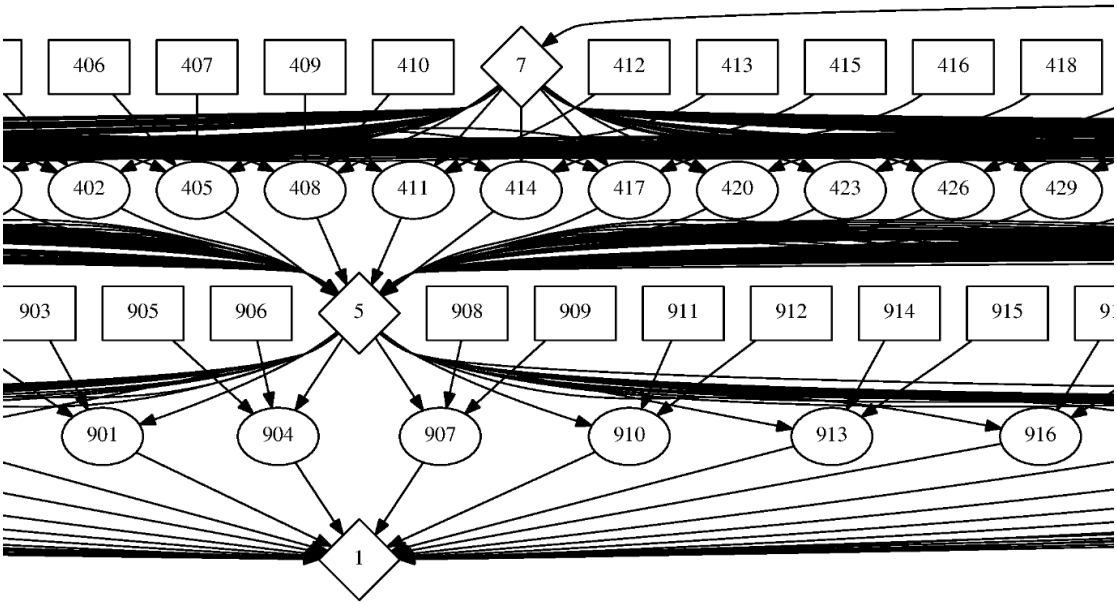

Figure 13. A Portion of the ISE Server Initial Attack Graph

Above, the screenshot shows multiple vertices that produce the possible paths for the attacks that can ultimately lead to running arbitrary code in the ISE Server with administrative rights. Each vertex was logically deduced by MulVAL engine from the Nessus assessment result.

Table 4 lists only a small portion of the vertices depicted in the Figure 13 graph.

Table 4. Attack Graph Vertices

1	execCode(iseServer,root)	OR
2	RULE 0 (local exploit)	AND
3	cvss('CVE-2003-0350',l)	LEAF
4	vulExists(iseServer,'CVE-2003-0350',windows_2000,localExploit,privEscalation)	LEAF
5	execCode(iseServer,someUser)	OR
6	RULE 1 (remote exploit of a server program)	AND
7	netAccess(iseServer,tcp,'445')	OR
8	RULE 5 (direct network access)	AND
9	hacl(hmn,iseServer,tcp,'445')	LEAF
10	attackerLocated(hmn)	LEAF

4.6 Quantitative Risk Analysis

The MulVAL's quantitative risk assessment algorithm, based on Wang et al. (2008), combines the CVSS metrics existing in the discovered vulnerabilities with the attack graph to calculate probabilistic risk metrics for the environment network. The script for the quantitative analysis requires the input files to contain summarized information, either summ_nessus or summ_oval for Nessus and OVAL XML formats, respectively, as it will always perform grouping.

A quantitative risk assessment for the construction was calculated based on the attack graph and CVSS scoring which was then presented to the NOC in parallel to the other rendered attack graphs.

Figure 14 – although still greatly reduced in size – shows a portion of a risk assessed attack graph for the environment. The use of grouping algorithms, especially vulnerability based grouping, clearly enhanced the graph's usability comparing to the one on Figure13.

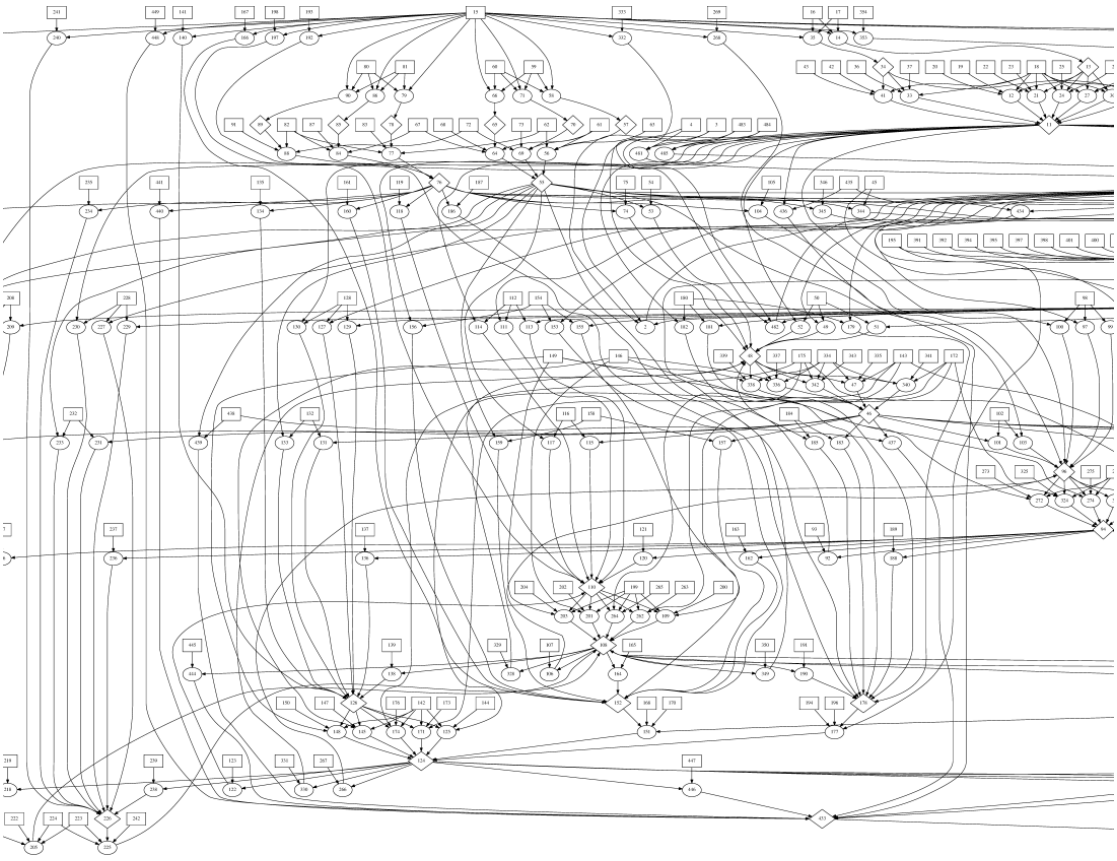

Figure 14. Attack Graph with Quantitative Risk Assessment

5 Use Case “HMN”

Not having a sponsor the thesis, the assessment for the attack graph integration could not be conducted in a real operational environment. Therefore, the usability assessment for the construction was carried out with a use case of a joint mission network where core services were published for the participating C2 systems to consume.

Harbinger Mission Network (HMN) is a fictitious deployable network-enabled-capability environment participating in and supporting joint command and control-lead operations. HMN is a platform for several information services that are actively advertised and published throughout the HMN network. C2 systems from participating nations can connect to the HMN to cooperate and collaborate in the joint missions, consuming the available services as well as publish their own set of services for other C2 systems to consume.

The security posture of the HMN was established from the assessment and configuration data of the construction in chapter 4.

5.1 HMN NOC Capabilities

5.1.1 Security Incident and Event Management (SIEM) System

The purpose of a Security Information and Event Management (SIEM) system is generally to provide centralized management for the collective log-based information and correlation of the log data with other information that is gathered through various types of data from their respective sources.

A SIEM system accepts information from numerous types of security related information sources, such as devices and sensors, network firewalls and IDS/IPS systems, host based IDSs, and is capable of performing normalization and correlation of the data it receives through these sources, to build a common view – a representation of the state of the environment to a SOC. SIEM will help the SOC to perform analytical and forensic investigations to the events that have occurred, it will

provide the SOC with tools that provide some level of threat intelligence, which together with an asset database, will build to the situational awareness of the network environment for an organization. Figure 15 depicts a typical SIEM architecture. (Bhatt et al. 2014)

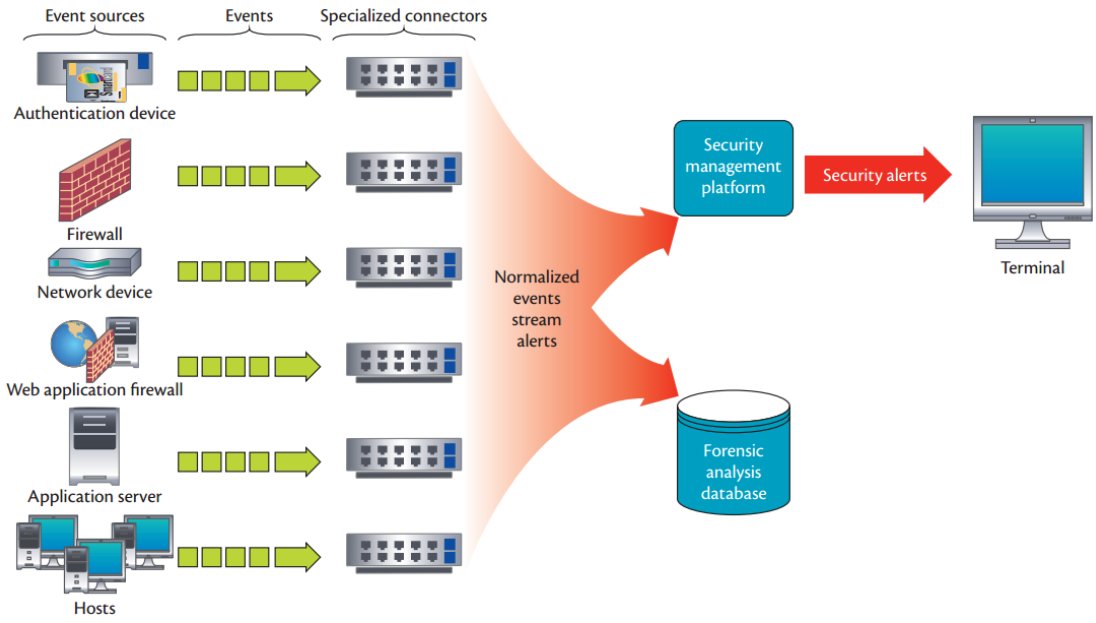

Figure 15. A Typical Security Incident and Event Management (SIEM) System Architecture. Quoted from Bhatt et al. (2014).

An Open Source SIEM from AlienVault Inc., OSSIM, was chosen as the SIEM system for HMN use in this thesis. OSSIM is an open source variant of the AlienVault's commercial SIEM product, USM. OSSIM contains the sensor and SIEM components for quick single-tier deployment, in which all the components are installed into a single server. For expansion, additional sensors can be later added and deployed into the environment. Additionally, OSSIM includes community-developed correlation rules for the threat analysis performed in the SIEM, with the ability to create customized correlation rules. (AlienVault®, OSSIM vs USM™).

Plugins are used in OSSIM sensors to extract data from logs produced by different data sources, which can then be used to create alarms into the SIEM dashboard. OSSIM comes with several built-in plugins for various log formats and provides the ability to build plugins for custom specific log formats (Alienvault® USM™ Deployment Guide).

Host-based Intrusion Detection System (HIDS) component in OSSIM watchguards the behavior and state of the Windows-based computer systems the HIDS agent is installed on by monitoring and collecting logs, detecting rootkits, monitoring file integrity and MS Windows registry integrity. Additionally, HIDS can be run in agentless mode in UNIX/Linux hosts and with network devices such as routers and firewalls to perform, for instance, configuration integrity checks (Alienvault® USM™ Deployment Guide).

Another feature with the HIDS component with OSSIM is active response that can be configured to launch applications and perform actions based on certain triggers (AV USM deployment guide)and can also be used to monitor network devices such as routers and firewalls and their configuration integrity with agentless operating mode (AlienVault® USM™ Deployment Guide).

In this thesis, OSSIM was used to 1) build an asset database of the environment, 2) run vulnerability assessments on the assets with the integrated OpenVAS scanner, 3) deploy and manage HIDS agents through management network and 4) monitor one network segment for intrusion signatures. For the latter, an additional OSSIM sensor was deployed as an IDS sensor. The deployed IDS was configured to use suricata as the IDS engine with a customized rule set that was based on the Emerging Threats Open rule set, included in OSSIM (AlienVault® OSSIM vs USM™).

5.1.2 Centralized Real-Time Logging System

One limitation with OSSIM was that the Alienvault logging component is available only for the commercial Alienvault USM product. For the construction, an open-source-based real-time logging system was built to provide the log analysis capability in the test network environment, to complement the SIEM system. Requirements for the logging components were 1) the ability to collect log messages from networks such that were 2) unstructured and that would be recorded at 3) high rate.

A centralized logging system was implemented with open-source components including syslog-ng-ose, Elasticsearch, Logstash and Kibana.

Syslog-ng was configured as the log collector capable of high-frequency operation, whose sole purpose was to catch all the log messages that were sent towards the logging component in the network. The components in the ELK stack (Elasticsearch, Logstash, Kibana) were used for structuring, analyzing and presentation of the log data. The logging system had the ability to receive any type of log data, and when a structure was required for analysis and presentation, it was possible to produce one's own parser for the log data.

5.1.3 Full Packet Capture and Analysis

In the HMN network environment, a full packet capturing and indexing analysis was implemented as a capability for the NOC to perform analysis over the network traffic. The analysis was possible once the initial vulnerability assessment had been completed and when both the host-based access lists and the set of required services had been studied, so that the NOC was able to establish a baseline for the network traffic that was considered normal in the HMN. The actual traffic record and metadata index analysis was carried out with a KVM VM instance that had the traffic capture and indexing service installed, and with a network interface dedicated to recording the traffic.

Normally, full packet capture would require significant resources just to store the network traffic data and a high bandwidth capable tap device. In the HMN use cases, however, the traffic was very light, which meant that the utilization of the hardware was minimal and that the limited dedicated resources were sufficient.

5.1.4 Vulnerability Feed Update over an Air Gap

Since the network environment did not have internet connection – direct or proxy – the vulnerability definitions were not automatically updated. To manually update the vulnerability definitions into MulVAL analysis engine while still maintaining the cross-domain principle, a unidirectional, one-way data transfer was implemented.

Unidirectional data transfer means the ability to limit network data to flow only in one direction. It is often achieved with information exchange gateways and so-called

data diodes in high security implementations involving industrial control systems (Jeon & Na, 2016), or components of critical infrastructure, where there are typically several security level areas and domains

The most common form of a data diode is an optical link, in which the physical structure of the transmitting laser emitter and the receiving laser detector allows the data to pass through in one-way only (Barker & Cheese, 2012).

A network tap device can be installed between two network devices, switches, firewalls or routers as a secure way to connect a network flow monitoring tool to the network. Similarly to the data diodes, the traffic is allowed to pass one-way only. In tap devices using copper medium, this is accomplished with a break in the path between physical Ethernet interfaces' and the Medium Access Controller (MAC) (Ixia White Paper, 2014).

Typically, network tap devices can be configured into aggregating or non-aggregating modes. The aggregating mode enables the device to combine the two full duplex network streams of data into a single monitor output. In non-aggregating mode, the traffic flow from either of the two devices is copied onto a single monitor output (Datacom Systems, Network Taps). Conceptually, this mode of operation enables one-way data flow. Figure 16 depicts the functional design of the network tap device used in the thesis network (Ixia White Paper, 2014):

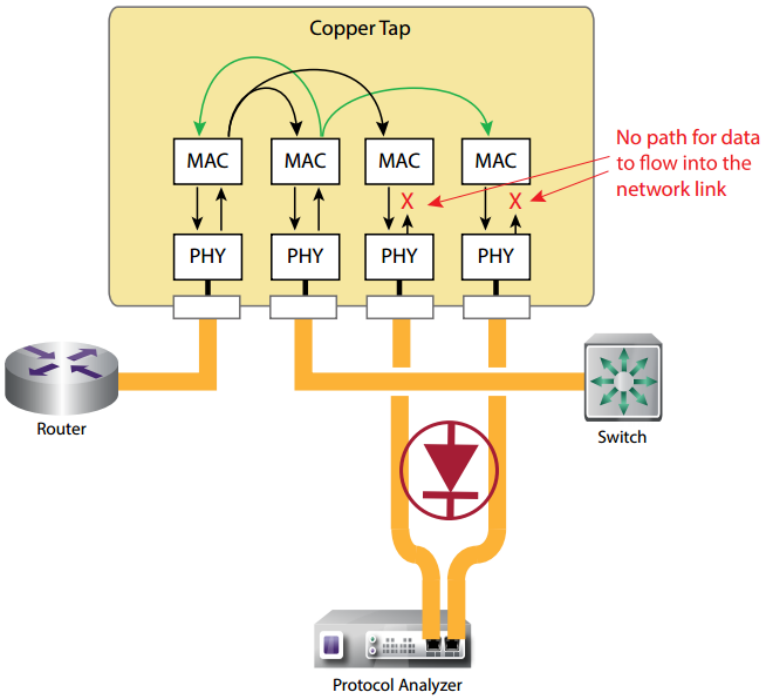

Figure 16. A Network Tap for Copper Medium. Quoted from IXIA White Paper (2014).

Forward error correction mechanisms, such as the Reed-Solomon implementation used by Heo et al. (2016) in their security gateway design, are generally used to assure that the integrity of the data being transferred in unidirectional applications will remain intact. Some implementations such as the Feedback Node, depicted by Jeon & Na (2016), have introduced the use of multiple data diodes for data transfer assurance, enabling retransmissions of the packets that were lost in transit.

Due to the simple structure of the network in this construction, a conceptually similar setup was achieved with using two raspberry pc's each having their host-based firewalls configured either to send or to receive, respectively, and a small program *udpcast* to handle one-way packet transfer during the manual NVD Feed vulnerability definitions update.

Udpcast is a program designed to broadcast data to multiple hosts simultaneously. It can be applied to a unidirectional data transfer as well, by instructing the program to perform a point-to-point asynchronous transmit.

Forward error correction in udpcast compensates for the packets lost in transit in a way, such that, for every S blocks of data there will be R number of redundant blocks and the data is divided among I stripes. Using the command below, *udpcast* was instructed to use unidirectional transfer with limited bitrate and a FEC with S=64, R=6 and I=8, which allowed for losing 48 subsequent packets and still be able to complete the transmission. (Udpcast Documentation)

```
udp-sender -f /opt/nvd_feed_data.tar.gz --async --
pointopoint -m 10.0.7.3 --max-bitrate 16k --fec 8x6/64
```

5.2 HMN Organizational structure

The fictitious organization for the HMN can be seen in Figure 17. The roles involved in the use case – ISSM and ISSO, highlighted in the figure – were tasked to watchguard and maintain the cyber security of the HMN with the support of the existing NOC capabilities.

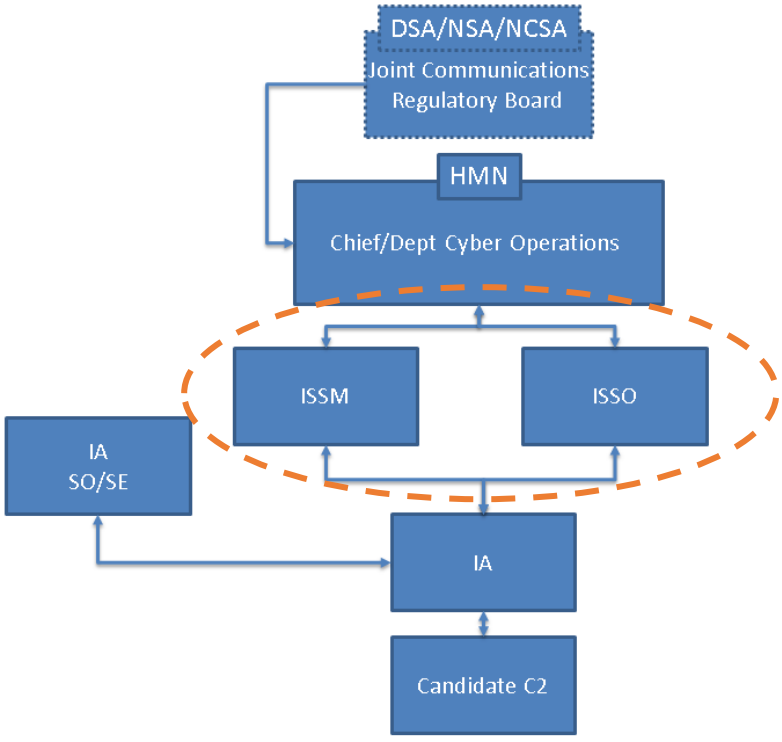

Figure 17. ISSM and ISSO Roles in the HMN Use Case

Ideally before connecting the candidate C2 system to the HMN network the C2 system would need to have an Infosec Assurance liaison assigned who, together with a Security Officer (SO) or a Security Engineer (SE), would help the ISSM and ISSO to establish an initial state, a baseline, of the candidate system's security posture. This would include the vulnerability and configuration assessments and provide ISSM with a view on the possible threat increase against HMN through the candidate C2 system. In this use case however, such process was not mandatory for the candidate C2 systems and their security posture remained unknown. Existence of possible malware and sub-optimal configuration was simulated through the attack cases in the following chapters.

5.3 Recognized Threats in HMN

Following threats regarding candidate C2 systems were recognized, such that could endanger the HMN. The list is artificially limited for the brevity of this use case.

1. Known malware is introduced to the HMN environment through a candidate C2
2. Unknown malware is introduced through candidate C2 and is able to persist in the HMN environment
3. Administrator exposes vulnerabilities by committing configuration errors or through poor management of the candidate C2
4. Devices that connect to HMN are not known due to not having enforced a strict device policy
5. Uncontrolled data exfiltration through newly connected devices and out-of-band communication channels
6. Uncontrolled candidate C2 system modification, changing services behavior and possibly allowing for unauthorized access
7. Undetected system access due to missing or falsely configured access control

5.4 Planning the Response

Regardless of not having implemented host valuation or other ISMS (ISO 27001, 27002 and 27005) risk management methods or controls in the construction, ISE

server, RADIUS server and LOG servers were designated as the most critical hosts in the environment.

Mitigating 0-day attacks

The conventional vulnerability assessment conducted with scanners such as those listed in chapter 3.3 will only detect known vulnerabilities. In the context of the HMN network, the most interesting vulnerabilities became those of which there is no previous knowledge.

In the absence of a sponsor, the operational aspect of the decision making process was not included with regard to tasking the NOC. Instead, the functionality of NOC was purely technical. As reactive and proactive countermeasures, the NOC decided to use methods: 1) reactive firewall rules; 2) full packet capture; and 3) revertation scripts of the planned counteractions from chapter 2.2.4.

Referred to as 0-day or Zero-day vulnerabilities, for instance, by Bilge & Dumitras (2012), Wang et al. (2010), Patel & Thaker (2011) and Zhang et al. (2011), are the kind of vulnerabilities that are not yet known to the public. Companies that are specializing in discovering 0-day vulnerabilities and such that are capable of supplying them on an annual basis even have subscription service models for their exploitation (Herr, 2016), and have also been reported selling the 0-day details to companies and governments (Fung, 2013).

According to Hutchins et al. (2010), even the use of zero-day exploits by the malefactors may be discovered if they are delivered or exploited using a method that has been used before. Hutchins et al. (2010) suggest that the ability to revisit the attacks and reconstruct intrusions would be particularly useful. Being able to recognize patterns or signatures of unorthodox and advanced methods could effectively prevent their reuse and would likely increase the required cost of the malefactors' campaigns. (Hutchins et al. 2010).

In HMN, NOC decided to plan for the mitigative actions against 0-day attacks using the configuration information from chapter 4 as the starting point. NOC gathered that the vulnerability scanners used in those assessments were of little use.

Additionally, NOC used MulVAL to generate probabilistic attack graph and implemented reverse logic into the decision making process. In other words, NOC decided that the multistage attack that seemed as the most likely attack paths were, in fact, the least probable the malefactors would utilize. NOC concluded that the malefactors would unlikely risk being seen by using an exploit to a known vulnerability as there might be signature for the exploit available.

NOC used the HMN attack graph in conjunction with the configuration information and opted for hardening the core services as thoroughly as possible, following some of the models and guidelines of the *k*-zero day safety, by Wang et al (2010).

5.5 Attack Cases

5.5.1 Unknown Malware

In this attack, a previously unknown malware had been introduced into HMN through a C2 system. The malware had been able to make its way to and infect the ISE server. It was unclear how the initial distribution of the encrypted or obfuscated binaries had been delivered to the ISE server.

The first indicator of the infection was spotted by NOC through the centralized logging system several days after they had deployed the HIDS file integrity component, sending its daily logs to the log server. The initial run for the malware and the delivery of the encrypted executables had to have happened before the HIDS deployment.

Once infected, the malware had launched a process with at least two threads. The first thread was actively – yet slowly – scanning for windows-based computers in the networks the server had access to. The second thread was used to deobfuscate or decrypt the previously planted 36 polymorphic copies of the program in order to create persistency and to allow for later remote commands through a reverse connection through port 445. Figure 18 shows the file operations for the ISE server where the initial indicator, the decryption phase of the malware on September 12th, is highlighted with the first vertical bar.

Once decrypted, the malware had produced several executable that were identical to the size of the first version that was encountered – 92447 bytes. The new polymorphic versions of the malware were named randomly, having eight random characters and 2 to 3-digit padding or suffix at the end.

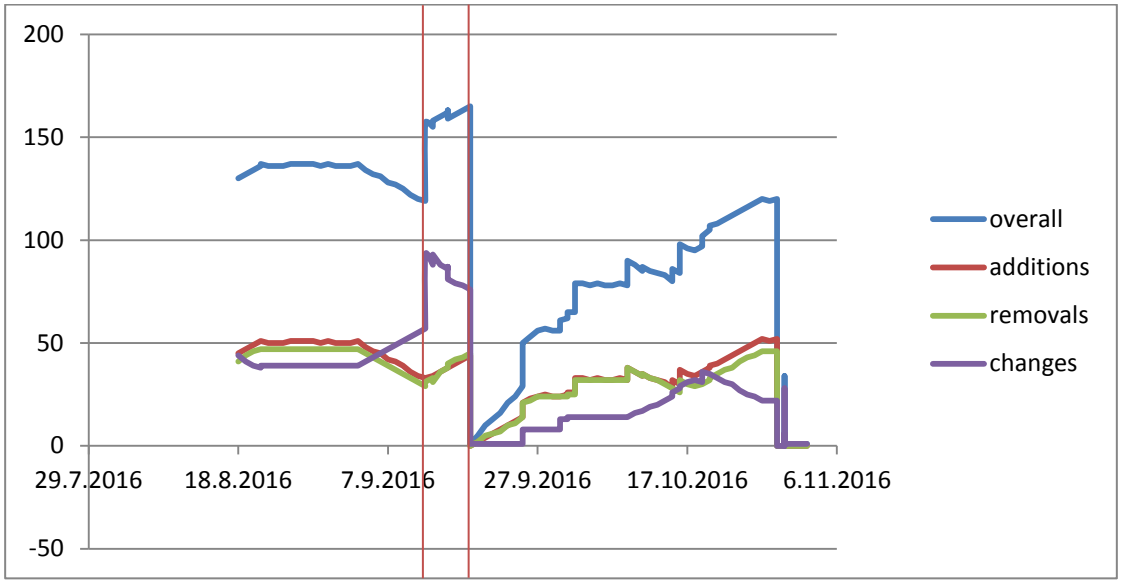

Figure 18. File Alterations of a Polymorphic Malware in ISEServer

After the indication NOC was quickly able to isolate the malware having learned its propagation attempts in the full packet capture metadata indexer.

The remediation actions performed by the NOC consisted of two phases, the effects of which can also be seen in the Figure 18 graph. The first remediation step included orchestrating an enforcement script, which effectively prevented the malware from altering files in the existing filesystem, after which the file integrity database was signed for approval, highlighted with the second vertical line in the graph timeline. The malware still persisted, but its functionality was now reduced. The second remediation step included finding and removing the main process, effectively disarming the malware in the ISE Server.

Since the time of initial infection was unknown, NOC created a manual SCAP-compatible OVAL-test (Figure 19) that would indicate the presence of the encountered malware in a Windows-based system. This information and identification signature was then distributed to the C2 that was connected to the HMN at the time of the malware discovery.

```
<objects>
  <registry_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:default.n
    <hive datatype="string">HKEY_CURRENT_USER</hive>
    <key datatype="string">Volatile Environment</key>
    <name datatype="string">HOMEDRIVE</name>
  </registry_object>
  <file_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:default.names
    <behaviors recurse_direction="down" max_depth="50" />
    <path datatype="string" var_ref="oval:default.namespace:var:1" var_check="all" />
    <filename datatype="string" operation="pattern match">[a-z]{1,8}[0-9]{1,3}.[eE][xX][eE]</filename>
  </file_object>
</objects>
<states>
  <file_state xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:default.namesp
    <size datatype="int" operation="equals">92447</size>
  </file_state>
```

Figure 19. An Excerpt of a Custom SCAP (OVAL) Test for the Polymorphic Malware

In a small environment such as the HMN, the unknown malware attack case demonstrated limited benefit from the attack graph analysis of the construction. The attack graph did show that a multipath combination was not possible within the HMN, which was quite obvious since no other Windows-based machine existed in the network. However, this was not immediately known to the NOC, not before the malware had been thoroughly investigated. This meant that until the dissection of the malware, NOC needed to take preventive measures to ensure the protection of the HMN assets and for this purpose the attack graph was well utilized.

5.5.2 Remote Connection Through Side Channel (ircd)

In this attack, the main utility used for the operational instant messaging system (chat) in the HMN environment contained a trojan, a backdoored binary through which a malefactor had been able to control the host machine with the same (`root`) privileges that the chat service daemon was running.

The backdoored version of the chat server was listening on the port 6667, which was flagged as legit traffic by the NOC team based on the initial service and vulnerability assessments. An attack graph was not available for the chat server, since the server was clear of vulnerabilities based on the assessment result and therefore no attack paths could be rendered.

When a C2 system was connected through node CR (named CRC2) to the HMN, and the routing information had been exchanged and configured, the network security engineer in charge of the CRC2 soon notified the NOC that they had seen traffic through their network towards the HMN chat server, even if they themselves had not yet configured their chat clients to use that server.

During the investigation NOC had the advantage of relatively accurate time window in which the event had occurred. The time window together with the relevant IP information from the CRC2 network engineer, NOC was rather quickly on top of the specific time of the event. In the SIEM system, they found nothing out of the ordinary in the specific timeframe and IDS logs.

In the packet capture metadata, on the other hand, the NOC was able to revisit the specific time window and they were able to go through the specific TCP session. NOC immediately applied a firewall rule to the perimeter firewall to prevent further connections to the destination IP that was discovered.

The NOC were able to discover that the server daemon was indeed backdoored, and that there was no patch available that would remediate its further exploitation. The HMN requirement that the instant messaging had to be available through TCP port 6667 forced the NOC to search for another form of mitigation.

Lacking any attack graph analysis for chat server NOC decided that isolating the attacker to just to the chat server required the host-based (iptables) firewall to prevent outgoing connections to the other network segments, which was then enforced with a SCAP XCCDF evaluation rule with forced remediation, resulting in an iptables rule being inserted into the OUTPUT rule chain.

This did not however, remove the remote access to the backdoored daemon. Knowing only one IP address from which the remote connection had been established was not sufficient. During the further investigation NOC was able to discover that the backdoor triggering packet contained letters “*QQS*”, which were not seen in any of the subsequent packets in the session data. The NOC then ended up implementing a remediation script, very specific to the particular backdoor that enforced an iptables rule to block incoming TCP packets through port 6667 that contained the signature “*QQS*”,

```
iptables -I INPUT 3 –i ens3 -m state --state NEW –p
tcp --dport 6667 –m string  --algo bm --string “QQS”
–j DROP
```

This effectively prevented the further exploitation of the backdoor regardless of the source IP address.

Similarly to the unknown malware attack case, NOC was unable to remove the binary that was backdoored and they had to implement mitigative actions such that increased the resiliency of the server, in which the backdoor persisted. In this attack case no attack graph analysis was readily available for the chat server since the vulnerability assessment did not find vulnerabilities in the server.

Since the MulVAL utility does allow for versatile customization, the NOC were able to create an attack graph for chat server for later use. In the Nessus translated input file they simply added a custom line describing the newly discovered backdoor and the privileges the daemon was being run with (`root`).

The new attack graph then added to the resilience of the chat server because after visualizing the new graph NOC noticed that they had only enforced the string based drop rule to the core network interface (ens3), but the backdoor could still be exploitable through management network. The NOC therefore expanded the firewall rule to also include the management network.

5.5.3 Data Exfiltration using ICMP echo requests and DNS requests

In this attack data from was being exfiltrated inside what appeared like ICMP ping requests and DNS queries towards an outside IP address over Side Channel connection. The NOC operator had not noticed anything out of the ordinary during the overwatch of the network traffic profile.

The data exfiltration was carried out using two phases: 1) first by sending the data to an undocumented core network IP address using ICMP protocol. A rogue device had been installed into the core network and was functioning as the receiving host. The

receiver compiled the received data and in turn started to send the data using DNS queries through a mobile access point uplink, the receiver was attached to.

A backdoor that existed in the chat server (attack case 5.5.2) had allowed for a remote access to the chat server with privileged rights. Even though the NOC had been successful in remediating the backdoor exploitation in 5.5.2 the attacker had been able to build a script that was slowly sending the chat server internal user database to a receiver in encoded and subsequent ICMP echo requests.

It seems the NOC was able to discover the ongoing exfiltration by chance. The SIEM system had discovered a new device during an automated asset sweep, but the discovery had been left unnoticed until the NOC was starting to perform a vulnerability assessment on another host.

Having only recently finished the chat server backdoor case, the discovery of a new device immediately triggered a thorough investigation during which the lateral ICMP requests were also discovered from within the packet capture indexer.

The echo requests were slowed down to not raise attention in the environment. The ICMP "*ping*" was performed every one second plus a randomized delay of 0 to 8 seconds. During the investigation NOC was able to discover the variance from within the centralized logging system, where luckily the chat server was configured to send all the kernel-level logs, including firewall logs. The discovered time variance is depicted in Figure 20.

The rogue device was disconnected from the network, but kept running to further examine the attack case. The device was sending data, assumingly the partially gathered user database, encrypted within DNS queries over UDP port 53.

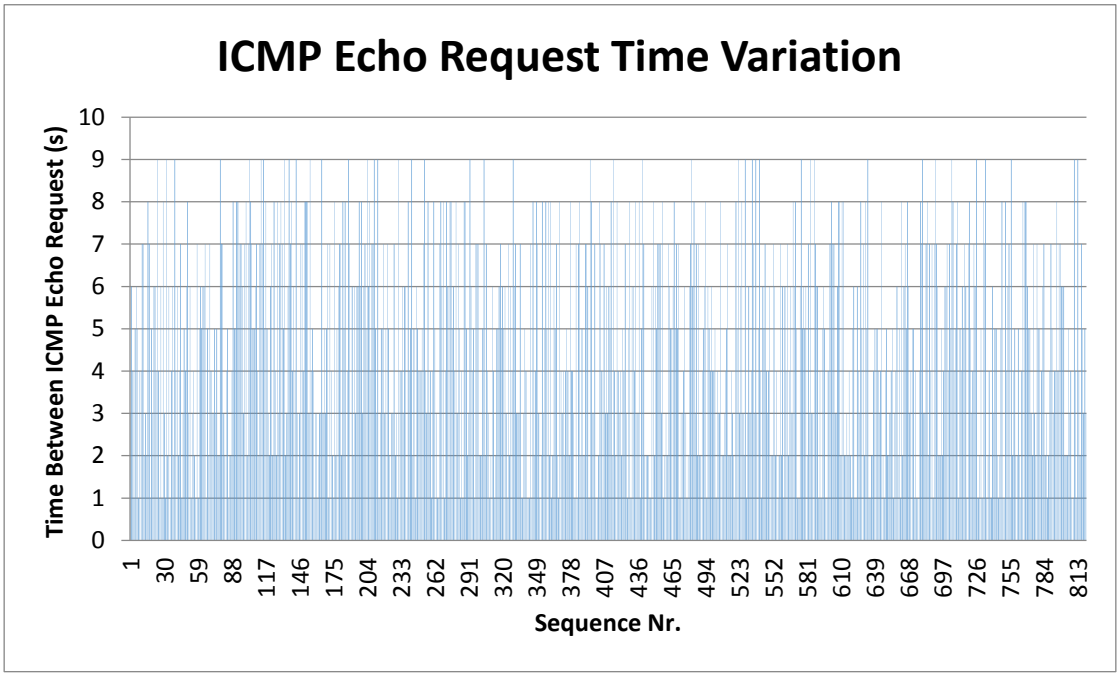

Figure 20. ICMP Exfiltration Time Variation

In this attack case, the attack graph provided little use to the NOC. New device insertion by an insider threat actor in this attack case was only detectable through the SIEM system.

The attack case did eventually improve the future resiliency of the HMN network, since after this attack case, NOC implemented audit logging on every host so that in the future, every console command would be securely sent do the centralized logging system which the NOC would be able to monitor for specific commands such that are common to malefactors when arriving at new, undiscovered hosts.

6 Conclusions

Taking into account that the construction and its usability assessments were in fact built and performed by the same individual, the conclusions are not entirely objective.

Despite not having the opportunity of a sponsor for the thesis and thus not being allowed to test the construction in a truly operational environment to enable course

of action planning, the construction was considered useful in the context of this research.

Through the construction the research was able to support a very small NOC – some might even refer to as a One-man SOC – in measuring the overall security of a limited set of hosts in a semi-isolated network. The integration of the attack graph analysis system was successful and the construction was able to provide additional information for the NOC operator with respect to the measurement and analysis of the static-like security posture of the semi-isolated environment, utilized particularly in the HMN use case. MulVAL provided the ability to analyze the overall security combining vulnerability information with the underlying service and network-level interdependencies, and to create visual presentation for the analysis.

Additionally, traditional procedures for maintaining software and security updates could benefit for the cyclic analysis of the data, although the processed data would likely be more useful to integrate as machine-readable .xml as opposed to a rendered visualization. That way the data would be more convenient to refine or restructure for various different uses, manual or automated.

During the attacks, the construction provided means to rapidly react to them, allowing for the actions to intercept lateral expansion or to cut before the next phase of the attack could take place.

Before the potential attacks, the most vulnerable assets were identified. The mitigation planning was possible to be conducted then based on the probabilistic analysis and quantitative risk assessment by MulVAL. Being able to visualize the most probable attack steps and to use that information to prioritize the mitigative actions was beneficial.

In the real-world operational networks, however, especially when the state-sponsored malefactor are involved, the attack graphs representing the most likely exploited vulnerabilities and the respective attack paths can be misleading. Normally, one would think the attackers would always be looking for the easiest and least-cost attack path in every environment and every situation. While in contrast, state-sponsored malefactors will likely build their strategy on reverse logic and decide that what would seem as the most sensible attack path is in fact unusable due to the

existence of the defender. Similarly, following the more expensive and resource consuming attack path could be selected as the most convenient strategy. This would render the most expensive attack path the most likely, which of course would create completely different attack graph visualization.

The use cases in this research were overall very simple, as was the environment they were tested in. Still, three of the servers running inside the HMN network contained so many known vulnerabilities that the rendered attack graph visualizations became challenging for the NOC to read, and they were practically unusable without applying MulVAL's grouping algorithms on them. Two of the servers contained practically no known vulnerabilities, so they didn't constitute to any of the graphs.

However, again in an operational environment, the unknown vulnerabilities were more interesting than the known vulnerabilities. While the two servers appeared having no vulnerabilities, they were still configured similarly to the other two that contained the highest number of the vulnerabilities. As with the known vulnerabilities, the underlying configuration could allow also for the exploitation of the currently unknown, zero-day vulnerabilities.

The construction proved useful in this regard as well: Even if the CVSS-scoring-based probabilistic attack graphs did not render for an unknown malware, the underlying dependencies resulting from the host and network configuration still allowed for basic attack graph block visualization in text format. Instead of weighing it with characteristics that didn't exist, the graph depicted the logical connections, which in turn allowed for mitigation planning against zero-day attacks. The practical use cases demonstrated the usability of orchestrated counteractions that were easily deployed through the Security Content Automation Protocol and the OpenSCAP utility.

OpenSCAP supported manually created checklists, which turned out as very efficient and resource preserving way of checking for malware in the HMN unknown malware use case in which new signature had to be created manually by the ISSO as the malware was unknown to any AV. Additionally, this specific use case simulated a situation involving a legacy system where such AV scanner was not possible to install.

OpenSCAP also allowed for the automated XCCDF compliance checks to be reported centrally to the logging system, and a deviation from a known-good configuration set was quickly detected.

The SIEM system of the construction provided another level of visibility over the environment, allowing for the NOC operator to supervise the network for signature-based intrusions and perform automated vulnerability scanning for the assets as well as discovery of new assets.

In parallel to MulVAL, another tool that was most valuable to the NOC was the full packet capture, indexing and analysis system that allowed the NOC to very effectively supervise all the sessions, protocols and ports, packets and databytes, even files that were being transferred in the network. It also provided the NOC operator the possibility to chronologically revise the network-utilizing attacks. Through the construction it was possible for the NOC to monitor the use of the allowed services also laterally. For instance in the HMN exfiltration attack case where the ICMP echo requests were not a violations per se, the incident was escalated by the NOC as an anomaly based on the full packet capture metadata indexing data.

The construction applied in this research provided methods for a NOC operator to protect a small operational environment and also to improve its security poster, resulting in a narrower and more confined threat landscape for the malefactors. In that regard, the work conducted in this research will likely assist in solving some of the real-world operational network problems in the future.

Decision makers involved in network-centric operations should possess at least moderate knowledge of how cyber operations are conducted especially with regard to state-sponsored actors' involvement, so that the resiliency of the cyber operations would not be as heavily depending on the availability of the subject matter experts. Automation of network-enabled defenses can help only so much, and conventional methods will still be required for resilience and persistence in the network-centric operations.

Lastly, the attack graph analysis contributed to the planning of some level of tactical and operational level countermeasures, which ideally could allow for the creation of unique and practical "*playbook*" for the cyber operation defense decision makers.

6.1 Areas of Future Research

To further develop the resiliency of operational networks, the many applications of Artificial Intelligence should be intensively studied especially with regard to autonomous systems in cyber defense and cyber offense. Dynamic learning abilities, self-healing and dynamically adapting networks and other autonomous capabilities leveraging artificial intelligence would be intriguing.

In their work, Alsaleh & Al-Shaer (2011) presented a framework combining regular SCAP-based host configuration compliance checks with network configuration analysis such as network path compliance, and transformed as logical objects to be presented as Prolog facts. Areas such as this, especially when utilizing the machine readable outputs for example for expert systems, learning systems, semi-autonomous defense systems could also be interesting.

References

Alienvault® USM™ Deployment Guide. Accessed on 18.10.2016. Retrieved from https://www.alienvault.com/documentation/resources/pdf/usm-deploymentguide.pdf.

AlienVault®, OSSIM vs USM™ whitepaper. Accessed on 18.10.2016. Retrieved from https://www.alienvault.com/resource-center/white-papers/ossim-vs-usm.

Alsaleh, M. N & Al-Shaer, E. SCAP Based Configuration Analytics for Comprehensive Compliance Checking. In proceedings on the 4th Symposium on Configuration Analytics and Automation (SAFECONFIG), 2011.

Argus CyberSecurity Lab, Kansas State University. 2012. MulVAL v1.1: A logic-based, enterprise network security analyzer. Source code. Retrieved from http://www.arguslab.org/mulval.html.

Asgarli, E. & Burger, E. 2016. Semantic Ontologies for Cyber Threat Sharing Standards. In proceedings on IEEE Symposium on Technologies for Homeland Security (HST).

Balabit PLC. 2016. The syslog-ng Open Source Edition 3.4 Administrator Guide, Accessed on 5.11.2016. Retrieved from https://www.balabit.com/documents/syslog-ng-ose-3.4-guides/en/syslog-ng-ose-guide-admin/pdf/syslog-ng-ose-guide-admin.pdf.

Barker, R.T. & Cheese, C.J. 2012. The Application of Data Diodes for Securely Connecting Nuclear Power Plant Safety Systems to the Corporate IT Network. 7th IET International System Safety Conference, incorporating the Cyber Security.

Bhatt, S; Manadhata, P.K; Zomlot, L. 2014. The Operational Role of Security Information and Event Management Systems. IEEE Security & Privacy, Volume: 12, Issue: 5, pp. 35-41.

Bilge, L and Dumitras, T. 2012. Before We Knew it: An Empirical Study of Zero-Day Attacks in the Real World. In proceedings on the 2012 ACM conference on Computer and communications security. pp 833-844.

Bratko, I. 2011. Prolog Programming for Artificial Intelligence. Harlow: Pearson Education Limited, Fourth Edition.

Casenove, M. 2015. Exfiltrations Using Polymorphic Blending Techniques: Analysis and Countermeasures. In Proceedings on the 7th International Conference on Cyber Conflict. pp. 217-230.

Casola, V; De Benedicts, A; Rak, M. 2015. Security Monitoring in the Cloud: An SLA-Based Approach. In proceedings on the 10th International Conference on Availability, Reliability and Security (ARES). pp. 749-755.

CERT Advisory. 1988. Computer Emergency Response Team, Carnegie Mellon University. CA-1988-01: ftpd Vulnerability. Accessed on 10.11.2014. Retrieved from http://www.cert.org/historical/advisories/CA-1988-01.cfm.

Colmerauer, A; Roussel, P. 1992. The birth of Prolog. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.7438&rep=rep1&type=pdf.

Conti, G; Nelson, J; Raymond, D. 2013. Towards a Cyber Common Operating Picture. In Proceedings on the 5th International Conference on Cyber Conflict. pp. 279-296.

CVE – Common Vulnerabilities and Exposures. About CVE. Accessed on 7.12.2015. Retrieved from https://cve.mitre.org/about/.

CVE – CVE ID Syntax Change. Updated 13.9.2016. Accessed on 9.10.2016. Retrieved from https://cve.mitre.org/cve/identifiers/syntaxchange.html.

CVSS – A Complete Guide to the Common Vulnerability Scoring System Version 2.0. FIRST.org, Inc. Accessed on 11.11.2015. Retrieved from http://www.first.org/cvss/v2/guide.

Dandurand, L; Serrano. O.S. 2013. Towards Improved Cyber Security Information Sharing. In Proceedings on the 5th International Conference on Cyber Conflict. pp. 9-25.

Datalog User Manual. 2004. Version 2.2. The MITRE Corporation. http://www.ccs.neu.edu/home/ramsdell/tools/datalog/datalog.html.

Datacom Systems Inc. Network Taps. Accessed on 16.10.2016. Retrieved from http://datacomsystems.com/products/network-taps.

Daud, N.I; Bakar, K.A.A; Hasan, M.S. 2014. A Case Study on Web Application Vulnerability Scanning Tools. In proceedings on the Science and Information Conference (SAI). pp. 595-600.

Dodig-Crnkovic, G. 2010. Constructive Research and Info-Computational Knowledge Generation In Model-Based Reasoning in Science and Technology. Studies in Computational Intelligence. Volume 314. pp 359-380.

Dressler, J, Moody, W, Bowen, Calvert L. III; Koepke, J. 2014. Operational Data Classes for Establishing Situational Awareness in Cyberspace. In Proceedings on the 6th International Conference on Cyber Conflict. pp.175-186.

FireEye®. 2014. Special Report APT28: A Window into Russia's Cyber Espionage Operations? Accessed on 2.5.2016. Retrieved from https://www2.fireeye.com/Services_Campaign_APT28_EMEA.html.

FireEye®. 2015. Special Report APT29: Hammertoss: Stealthy Tactics Define a Russian Cyber Threat Group. Accessed on 2.5.2016. Retrieved from https://www2.fireeye.com/rs/848-DID-242/iamges/rpt-apt-29-hammertoss.pdf.

FireEye®. 2015. Special Report APT30 and the Mechanics of a Long-Running Cyber Espionage Operation. Accessed on 2.5.2016. Retrieved from http://www2.fireeye.com/rs/fireye/images/rpt-apt30.pdf.

F-Secure® Threat Report. 2015. Accessed on 2.5.2016. Retrieved from https://secure.f-secure.com/threat_report.html.

Fung, Brian. 31.8.2013. The NSA hacks other countries by buying millions of dollars' worth of computer vulnerabilities. The Washington Post article. Accessed on 22.10.2016. Retrieved from https://www.washingtonpost.com/news/the-switch/wp/2013/08/31/the-nsa-hacks-other-countries-by-buying-millions-of-dollars-worth-of-computer-vulnerabilities/.

Gadelrab, M, S & Ghorbani, A. 2012. A New Framework for Publishing and Sharing Network and Security Datasets. In SC Companion: High Performance Computing, Networking, Storafe and Analysis (SCC). pp. 539-546.

Gallon, L; Bascou, Jean-Jacques. 2011. CVSS Attack Graphs. In Proceedings of the 7th International Conference on Signal Image Technology & Internet-Based Systems. pp. 24-31.

Guarino, A. 2013. Autonomous Intelligent Agents in Cyber Offence. In Proceedings on the 5th International Conference on Cyber Conflict. pp. 377-389.

Hammersley, B. 4.2.2015. Why you should be a e-resident of Estonia. The Wired Magazine article. Accessed on 6.7.2016. Retrieved from http://www.wired.co.uk/article/estonia-e-resident.

Haaster, Jelle van. 2016. Assessing Cyber Power. In Proceedings on the 8th International Conference on Cyber Conflict. pp. 7-22.

Heinl, Caitríona H. 2014. Artificial (Intelligent) Agents and Active Cyber Defence: Policy Implications. In Proceedings on the 6th International Conference on Cyber Conflict. pp. 53-67.

Heo, Y; Kim, B; Kang, D; Na, J. 2016. A Design of Unidirectional Security Gateway for Enforcement Reliability and Security of Transmission Data in Industrial Control Systems. In proceedings on the 8th International Conference on Advanced Communication Technology (ICACT). pp. 310-313.

Hernandez-Ardieta, J.L; Tapiador, J.E; Suarez-Tangil, G. 2013. Information Sharing Models for Cooperative Cyber Defence. In Proceedings on the 5th International Conference on Cyber Conflict. pp. 63-91.

Herr, T. 2016. Malware Counter-Proliferation and the Wassenaar Arrangement. In Proceedings on the 8th International Conference on Cyber Conflict. pp. 175-190.

Hlyne, C. N. N; Zavarsky, P; Butakov, S. 2015. SCAP Benchmark for Cisco Router Security Configuration Compliance. In proceedings on the 10th International Conference for Internet Technology and Secured Transactions (ICITST). pp. 270-176.

Hutchins, E. M; Cloppert, M. J; Amin, R. M. 2016. Intelligence-Driven Computer Network Dedense Informed by Analysis of Adversart Campaigns and Intrusion Kill Chains. Accessed on 18.5.2016. Retrieved from http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf

Im, Sun-young; Shin, Seung-Hun; Ryu, K.Y. 2016. Performance Evaluation of Network Scanning Tools with Operation of Firewall. In proceedings on the 8th International Conference on Ubiquitious and Future Networks (ICUFIN). pp.876-881.

Ingols, K; Chu, M; Lippmann, R; Webster, S; Boyer, S. 2009. Modeling Modern Network Attacks and Countermeasures Using Attack Graphs. In Computer Security Applications Conference. pp. 117 -126.

Introduction to Making Security Measurable. The MITRE Corporation. Accessed on 3.3.2016. Retrieved from https://makingsecuritymeasurable.mitre.org/about/index.html.

ISO/IEC 13211-1:1995. Information Technology – Programming Languages – Prolog – Part 1: General Core. April 20th 1995.

Ixia. 2014. Secure, Unidirectional Data Flow with Network Taps. White Paper 915-6894-01 Rev. A. Accessed on 16.10.2016. Retrieved from https://www.ixiacom.com/sites/default/files/resources/whitepaper/915-6894-01-secure-unidirectional-data_flow-network-taps.pdf.

Jakobson, G. 2013. Mission-Centricity in Cyber Security: Architecting Cyber Attack Resilient Missions. In Proceedings on the 5th International Conference on Cyber Conflict. pp. 357-375.

Jeon, Boo-Sun & Na, Jung-Chan. 2016. A Study of Cyber Security Policy in Industrial Control System Using Data Diodes. In proceedings on 8th International Conference on Advanced Communication Technology (ICACT). pp. 314-317.

Kalutarage, Harsha K; Shaikh, Siraj A; Zhou, Q; James, Anne E. 2012. Sensing for Suspicion at Scale: A Bayesian Approach for Cyber Conflict Attribution and Reasoning. In Proceedings on the 4th International Conference on Cyber Conflict. pp. 393-412.

Kamhoua, C; Martin, A; Tosh, D; Kwiat, Kevin A; Heitzenrater, C; Sengupta, S. 2015. Cyber-Threats Information Sharing in Cloud Computing: A Game Theoretic Approach. In proceedings on the 2nd International Conference on Cyber Security and Cloud Computing (CSCloud). pp. 382-389.

Kaur, R; Singh, M. 2014. Efficient hybrid technique for detecting zero-day polymorphic worms. In proceedings on the 2014 IEEE International Advance Computing Conference (IACC). pp.95-100.

Koch, R; Golling, M. 2013. Architecture for Evaluating and Correlating NIDS in Real-World Networks. In Proceedings on the 5th International Conference on Cyber Conflict. 335-355.

Koch, R; Golling, M. 2016. Weapon Systems and Cyber Security – A Challenging Union. In Proceedings on the 8th International Conference on Cyber Conflict. pp. 191-204.

Koch, R; Rodosek, G.D. 2013. The Role of COTS Products for High Security Systems. In Proceedings on the 4th International Conference on Cyber Conflict. pp. 413-427.

Koch, R. 2011. Towards Next-Generation Intrusion Detection. In Proceedings on the 3rd International Conference on Cyber Conflict. pp.151-168.

Kornmaier, A; Jaouën, F. 2014. Beyond technical data – a more comprehensive Situational Awareness fed by available Intelligence Information. In Proceedings on the 6th International Conference on Cyber Conflict. pp. 139-155.

Kotenko, I, Chechulin, A. 2013. A Cyber Attack Modeling and Impact Assessment Framework. In Proceedings on the 5th International Conference on Cyber Conflict. pp.119-143.

Labro, E. and Tuomela, T-S. 2003. On bringing more action into management accounting research: process considerations based on two constructive case studies. European Accounting Review. Vol. 12 No. 3. pp. 409-42.

Linda, O; Vollmer, T; Manic, M. 2009. Neural Network Based Intrusion Detection System for Critical Infrastructures. In 2009 International Conference on Neural Networks.

Lu, L; Safavi-Naini, R; Hagenbuchner, M; Susilo, W; Horton, J. 2009. Ranking Attack Graphs with Graph Neural Networks. In The 5th Information Security Practices and Experience Conference.

Lukka, K. 2000. The Key Issues of Applying the Constructive Approach to Field Research. In Reponen, T. (ed.). Management Expertise for the New Millenium. In Commemoration of the 50th Anniversary of the Turku School of Economics and Business Administration. Publications of the Turku School of Economics and Business Administration, Series A-1:2000. pp.113-128.

Maconachy, V. W; Schou, C. D; Ragsdale D; and Welch, D. 2001. A Model for Information Assurance: An Intergrated Approach. In Proceedings of the 2001 IEEE Workshop on Information Assurance and Security. United States Military Academy. Accessed on 3.2.2015. Retrieved from http://grothoff.org/christian/teaching/2007/3704/w2c3.pdf

Mandiant® M-Trends. 2015. A View from the Front Lines. Accessed on 2.5.2016. Retrieved from https://www2.fireeye.com/rs/fireye/images/rpt-m-trends-2015.pdf.

Mandiant® APT1. 2013. Exposing One of China's Cyber Espionage Units. Accessed on 2.5.2016. Retrieved from https://intelreport.mandiant.com/Mandiant_APT1_Report.pdf.

Marchetti, M; Pierazzi F; Guido, A; Colajanni, M. 2016. Countering Advanced Persistent Threats through Security Intelligence and Big Data Analytics. In Proceedings on the 8th International Conference on Cyber Conflictp. pp. 243-262.

Mell, P; Scarfone, K; Romanosky, S. 2007. A Complete Guide to the Common Vulnerability Scoring System Version 2.0. Accessed on 16.11.2014. Retrieved from https://www.first.org/cvss/v2/guide.

Mepham, K; Ghinea, G; Louvieris, P; Clewley, N. 2014. Dynamic Cyber-Incident Response. In Proceedings on the 6th International Conference on Cyber Conflict. pp. 121-137.

Mohammed, M.M.Z.E.; Chan, H.A; Ventura, N.; Pathan, A-S.K. 2013. An Automated Signature Generation Method for Zero-Day Polymorphic Worms Based on Multilayer

Perceptron Model. In proceedings on the International Conference on Advanced Computer Science Applications and Technologies. pp 450-455.

Morris, R.T. 928 F.2D 504. 1991. Decision. United States Court of Appeals. Accessed on 18.11.2014. Retrieved from http://morrisworm.larrymcelhiney.com/morris_appeal.txt.

Mulazzani, F; Sarcia, Salvatore A. 2011. Cyber Security on Military Deployed Networks. In Proceedings on the 3rd International Conference on Cyber Conflict. pp. 13-28.

Munir, R; Disso, J.P; Awan, I; Mufti, M. R. 2013. A Quantitative Measure of the Security Risk Level of Enterprise Networks. In proceedings on the 8th International Conference on Broadband and Wireless Computing, Communications and Applications (BWCCA). pp. 437-442.

Nessus Compliance Checks – Tenable Network Security. Accessed on 12.10.2016. Retrieved from https://support.tenable.com/support-center/nessus_compliance_checks.pdf.

Nessus v6 SCAP Assessment – Tenable Network Security. Accessed on 21.10.2016. Retrieved from http://static.tenable.com/documentation/Nessus_v6_SCAP_Assessments.pdf.

Nessus® Home. Accessed on 8.7.2016. Retrieved from http://www.tenable.com/products/nessus-home.

Nessus Professional – The Most Widely-Deployed Vulnerability Assessment Solution. Accessed on 8.7.2016. Retrieved from http://www.tenable.com/products/nessus-vulnerability-scanner/nessus-professional.

Nessus Plugins. Accessed on 8.7.2016. Retrieved from http://www.tenable.com/plugins/.

Nexpose. Accessed on 8.7.2016. Retrieved from https://www.rapid7.com/products/nexpose/

NIST National Vulnerability Database, NVD. NVD Home. Accessed on 6.7.2016. Retrieved from https://nvd.nist.gov/.

NIST National Vulnerability Database, NVD. Data Feed. Accessed on 6.7.2016. Retrieved from https://nvd.nist.gov/download.cfm.

OpenSCAP. OpenSCAP Features. Accessed on 8.10.2016. Retrieved from https://www.open-scap.org/features/.

OpenSCAP User Manual, version 1.0. Accessed on 2.9.2016. Retrieved from https://static.open-scap.org/openscap-1.0/oscap_user_manual.html.

OpenVAS. Open Vulnerability Assessment System, About OpenVAS. Accessed on 8.7.2016. Retrieved from http://www.openvas.org/about.html.

OpenVAS Architecture Overview. Accessed on 8.7.2016. Retrieved from http://www.openvas.org/software.html#architecture_overview.

Ou, X; Boyer, Wayne F; McQueen, Miles A. 2006. A Scalable Approach to Attack Graph Generation. In proceedings of the 13th ACM CCS Conference. pp 336-345.

Ou, X; Govindavajhala, S; Appel, A.W. 2005. MulVAL: A Logic-based Network Security Analyzer. 14th USENIX Security Symposium.

OVAL®. Open Vulnerability and Assessment language. The MITRE Corporation, About OVAL. Accessed on 9.1.2015. Retrieved from https://oval.mitre.org/about.

Patel, R and Thaker, C. 2011. Zero-Day Attack Signatures Detection Using Honeypot. In proceedings on International Conference on Computer Communication and Networks. pp 79-85.

Pfenning, Frank. 2007. Logic Programming, Lecture Notes, Carnegie Mellon University, Pennsylvania. Retrieved from http://www.cs.cmu.edu/~fp/courses/lp/index.html.

Raymond, D; Conti, G; Cross, T; Fanelli, R. 2013. A Control Measure Framework to Limit Collateral Damage and Propagation of Cyber Weapons. In Proceedings on the 5th International Conference on Cyber Conflict. pp. 181-197.

Risk Based Security. 2015. CVE/NVD: The High Price of 'Free'. Accessed on 12.7.2016. Retrieved from https://www.riskbasedsecurity.com/reports/CVE%20&%20NVD%20-%20The%20High%20Price%20Of%20Free.pdf

Security Content Automation Protocol. SCAP. Accessed on 2.9.2016. Retrieved from http://scap.nist.gov/index.html.

SCAP Specifications. SCAP 1.0. Accessed on 9.10.2016. Retrieved from https://scap.nist.gov/revision/1.0/index.html.

Smaill, Alan. 2015. Logic Programming: Semester 1, Lecture Notes. University of Edinburgh. http://www.inf.ed.ac.uk/teaching/courses/lp.

Spafford, Eugene H. 1988. The Internet Worm Program: An Analysis. Purdue University Technical Report. Accessed on 12.7.2016. Retrieved from http://spaf.cerias.purdue.edu/tech-reps/823.pdf.

Tapscott, D; 1997. The digital economy: promise and peril in the age of networked intelligence. New York: McGraw-Hill.

Tosh, D; Sengupta, S; Kamhoua, C; Kwiat, K; Martin, A. 2015. An Evolutionary Game-theoretic Framework for Cyber-threat Information Sharing. In proceedings on the IEEE International Conference on Communications (ICC). pp. 7431-7346.

Tyugu, E. 2011. Artificial Intelligence in Cyber Defence. In Proceedings on the 3rd International Conference on Cyber Conflict. pp. 95-106.

Tyugu, E. 2012. Command and Control of Cyber Weapons. In Proceedings on the 4th International Conference on Cyber Conflict. pp. 333-343.

Udpcast. Introduction. Accessed on 16.10.2016. Retrieved from https://www.udpcast.linux.lu/.

Wang, L; Jajodia, S; Singhal, A. 2007. Measuring the Overall Security of Network Configurations Using Attack Graphs. In proceedings on the 21st Annual IFIP WG 11.3

Working Conference on Data and Applications Security. Redondo Beach, CA. USA. pp 98-112.

Wang, L; Islam, T; Long, T; Singhal, A; Jajodia, S. 2008. An Attack Graph-based Probabilistic Security Metric. In proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08). pp. 283-296.

Wang, L; Jajodia, S; Singhal, A. 2010. *k*-Zero Day Safety: Measuring the Security Risk of Networks against Unknown Attacks. In proceedings on the 15th European Symposium on Research in Computer Security (ESORICS 2010). Springer-Verlag Lecture Notes in Computer Science (LNCS). Vol. 6345. pp 573-587.

Wang, L; Jajodia, S; Singhal, A. Noel, S. 2010. Measuring Security Risks of Networks Using Attack Graphs. International Journal of Next-Generation Computing. Vol. 1, No. 1. pp 113-123.

XSB. A Logic Programming and Deductive Database System. Documentation, Volume 1. Accessed on 27.4.2015. Retrieved from http://xsb.sourceforge.net/manual1/manual1.pdf.

Yung-Yu, C; Zavarsky, P; Ruhl, R; Lindskog, D. 2011. Trend Analysis of the CVE for Software Vulnerability Management. In the IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE International Conference on Social Computing. Boston, MA. USA.

Zhang, S; Ou, X; Homer, J. 2011. Effective Network Vulnerability Assessment through Model Abstraction. In the 8th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA).

Zhang, S; Ou, X. 2011. README Documentation of the MulVAL system. MulVAL v1.1. Retrieved from http://www.arguslab.org/mulval.html.

Appendices

Appendix 1. Algorithm for Prolog's Question Answering Procedure, Bratko (2011).

procedure *execute* (*Program, GoalList, Success*);

Input arguments:

Program: list of clauses

GoalList: list of goals

Output argument:

Success: truth value; Success wil become true if GoalList is true with respect to Program

Local variables:

Goal: goal

OtherGoals: list of goals

Satisfied: truth value

MatchOK: truth value

Instant: instantiation of variables

H,H',B1,B1',...,Bn,Bn': goals

Auxiliary functions:

empty(L): returns true if *L* is the empty list

head(L): returns the first element of list *L*

tail(L): returns the rest of list *L*

append(L1,L2): appends list *L2* at the end of list *L1*

match(T1,T2,MatchOK,Instant): tries to match terms *T1* and *T2*; if succeeds then *MatchOK* is true and *Instant* is the corresponding instantiation of variables *substitute(Instant, Goals)*: substitutes variables in *Goals* according to instantiation *Instant*

begin

if *empty(GoalList)* then *Success* := *true*

else

begin

Goal := *head(GoalList);*

OtherGoals := *tail(GoalList);*

```
        Satisfied := false;
        while not Satisfied and "more clauses in the program" do
          begin
            Let next clause in Program be
                H :- B1,...,Bn.
            Construct a variant of this clause
                H' :- B1',...,Bn'.
            match(Goal,H',MatchOK,Instant);
            if MatchOK then
                begin
                  NewGoals := append([B1',...,Bn'],OtherGoals);
                  NewGoals := substitute(Instant,NewGoals);
                  execute(Program,NewGoals,Satisfied)
                end
          end;
        Success := Satisfied
    end
end;
```

Appendix 2. Prolog Execution Traces for Three Variations of The Program *link.P*

Appendix 3. Algorithm for Reachability-based Grouping, Zhang et al. (2011)

Input: A set of (reachTo(h), reachFrom(h)) for each host h in a subnet

Output: A hash map L, which maps a group label α to a list of hosts having the same reachability (reachTo and reachFrom).

1. $Lr \leftarrow \{\}$ {Lr is a set of triples (α, reachToSet, reachFromSet).}
2. Queue $Q \leftarrow$ all the hosts of the given subnet{
3. $L \leftarrow$ empty map {initialize the return value}
4. **while** Q is not empty **do**
   $n \leftarrow$ dequeue(Q)
   if Lr contains (α, reachTo(n), reachFrom(n)) **then**
   $L[\alpha] \leftarrow L[\alpha] \cup \{n\}$ {if the reachability of n is the same as some other host that has been processed, add n to its equivalent class.}
   else
   create a fresh α
   $Lr \leftarrow Lr \cup (\alpha$, reachTo(n), reachFrom(n)) {Otherwise put its reachability information into Lr}
   $L[\alpha] \leftarrow \{n\}$
   end if
   end while
5. return L

Appendix 4. Algorithm for Vulnerability Grouping, Zhang et al. (2011)

Input: A set of ungrouped vulnerabilities on a machine (S_u)

Output: A hash map L that maps an application to its vulnerability score

1. $Lr \leftarrow \{\}$ {Lr is a set of applications that have appeared so far}
2. $L \leftarrow$ *empty hash map*
3. **while** $S_u \neq \{\}$ **do**
   take v from S_u
   if Lr contains (v.application) **then**
   if L[v.application] < $v.score$ **then**
   L[v.application] = $v.score$
   end if
   else
   L[v.application] = $v.score$
   Lr.add(v.application)
   end if
   end while
4. return L

Appendix 5. Algorithm for Configuration-based Grouping, Zhang et al. (2011)

Input: set τ containing all the *TraceStep* terms,
attacker's goal G
Output: logical attack graph (N_r, N_p, N_d, E, L, G).

1. $N_r, N_p, N_d, E, L \leftarrow 0$
2. For each $t \in \tau$ {
   let $t = because(interactionRule, Fact, Conjunct)$
3. Create a derivation node r
   $N_r \leftarrow N_r \cup \{r\}$
   $L \leftarrow L \cup \{r \rightarrow interactionRule\}$
4. Look up $n \in N_d$ such that $L(n) = Fact$,
5. If such n does not exist
   {
   create a new fact node n
   $L \leftarrow L \cup \{n \rightarrow Fact\}$
   $N_d \leftarrow N_d \cup \{n\}$
   }
6. $E \leftarrow E \cup \{(n, r)\}$
7. For each fact f in $Conjunct$ {
8. Look up fact node $c \in (N_p \cup N_d)$ such that
   $L(c) = f$,
9. If such c does not exist
   {
   create a new fact node c
   $L \leftarrow L \cup \{c \rightarrow f\}$
   If f is primitive { $N_p \leftarrow N_p \cup \{c\}$}
   else { $N_d \leftarrow N_d \cup \{c\}$}
   }
10. $E \leftarrow E \cup \{(r, c)\}$
    }
    }