



OULUN YLIOPISTO
UNIVERSITY of OULU

DEGREE PROGRAMME IN COMPUTER SCIENCE AND ENGINEERING

Tiandu Zhao

**ADAPTIVE MULTI-HOST SECURITY MEASUREMENT
WITH THE HSIS**

Master's Thesis
Degree Programme in Computer Science and Engineering
March 2016

Zhao T. (2016) Adaptive multi-host security measurement with the HSIS. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 53 pp.

ABSTRACT

In order to implement correct security countermeasures, one must understand the current situation in the network. Gaining situational awareness supports the user in making proper decisions while facing security incidents, thus, it is crucial that security information is gathered from the host systems at the highest accuracy. In addition, the data gathering should correspond to the current needs that may vary from host to host.

This thesis presents a tool to remotely manage and conduct adaptive security measurements on multiple hosts. Moreover, the tool can be used to visualize the measurements in real time. The focus of this thesis is to introduce adaptive security measurements. The usability and the requirements of the tool are evaluated in three scenarios.

The adaptive aspect of the tool optimises the data gathering and resource usage for the changes in the host environment. In addition, the tool supports user-created security measurements to gather specific data from the host. Therefore, a more accurate image of the current situation can be obtained.

The tool provides the common functions of security-measuring instruments to aid users in the process of implementing new measurements. Increased security information gives users a better understanding of the current security level.

Keywords: situational awareness, security instrumentation, security measurement

Zhao T. (2016) Usean tietojärjestelmän adaptiivinen tietoturvamittaminen HSIS-työkalulla. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 53 s.

TIIVISTELMÄ

Täsmällisten vastatoimenpiteiden toteuttaminen edellyttää verkon nykytilanteen ymmärtämistä. Tilannekuvan saavuttaminen tukee käyttäjän päätöksentekoa tämän kohdatessa tietoturvaongelmia, joten on tärkeää, että tieto tietoturvasta on kerätty mahdollisimman tarkasti tietojärjestelmistä. Lisäksi tiedonkeruun tulisi vastata aina kyseisen tietojärjestelmän tarpeita.

Tässä diplomityössä esitellään työkalu, jonka avulla voi etähallita ja suorittaa adaptiivisia tietoturvamittauksia useissa tietojärjestelmissä. Työkalulla on myös mahdollista visualisoida tietoturvamittauksien tuloksia reaaliajassa. Diplomityö keskittyy työkalun avulla tehtävän adaptiivisen tietoturvamittauksen esittelyyn. Työkalun käytettävyys ja vaadittavien ominaisuuksien täytyminen testataan kolmessa erilaisessa skenaariossa.

Työkalun adaptiivinen toiminnallisuus optimoi tiedonkeruuta ja resurssien käyttöä muuttuvassa ympäristössä. Työkalu myös tukee käyttäjän luomia tietoturvamittauksia, jotka keräävät tiettyä tietoa järjestelmästä. Tarkoilla mittauksilla voidaan saada parempi kokonaiskuva nykyisestä tilanteesta.

Työkalu tarjoaa yhteisiä toimintoja tietoturvamittaukseen ja tukee käyttäjää uusien mittauksien toteuttamisessa. Lisääntynyt tieto tietoturvasta auttaa käyttäjiä ymmärtämään tämänhetkisen tietoturvatason.

Avainsanat: tilannekuva, tietoturvainstrumentointi, tietoturvamittaus

TABLE OF CONTENTS

ABSTRACT

TIIVISTELMÄ

TABLE OF CONTENTS

FOREWORD

ABBREVIATIONS

1.	INTRODUCTION.....	7
2.	BACKGROUND OF SECURITY MEASUREMENT AND TERMINOLOGY	8
2.1.	Cybersecurity	8
2.2.	Security engineering.....	9
2.3.	Security measurements.....	10
2.3.1.	Security measurement terminology.....	11
2.3.2.	Properties of security measures.....	14
2.4.	Situational awareness	15
2.5.	Similar solutions.....	16
3.	FRAMEWORK FOR ADAPTIVE SECURITY MEASUREMENT	18
3.1.	Overview of system instrumentation.....	18
3.2.	Requirements for the framework.....	20
3.3.	Conceptual architecture.....	22
4.	THE HOST-BASED SECURITY INSTRUMENTATION SYSTEM AND TESTING	27
4.1.	Concrete architecture.....	27
4.1.1.	Structure of instrument.....	27
4.1.2.	Structure of Manager.....	29
4.2.	Functioning of the HSIS.....	32
4.3.	Implementation testing.....	38
4.3.1.	Test case: security audit visualisation with the MVS.....	38
4.3.2.	Test case: adaptive auditing with an external tool	40
4.3.3.	Test case: real-time security monitoring	42
4.3.4.	Fulfilling the requirements	43
5.	DISCUSSION	45
5.1.	Comparison to other similar solutions	45
5.2.	Limitations	46
5.3.	Future work	46
6.	CONCLUSION	48
7.	REFERENCES.....	49

FOREWORD

The thesis is written at VTT Technical Research Centre of Finland in the Cybersecurity team in 2015 and 2016. The work is part of the SAICS Situational Awareness in Information and Cyber Security project (2014-2016) funded by the Academy of Finland and VTT Technical Research Centre of Finland.

I would like to thank Senior Research Scientist Antti Evesti from VTT for his great support and reviews. I also thank Professor Juha Rönning from the University of Oulu for his instructions and reviews of the work, and the second reviewer Doctor Thomas Schaberreiter for his reviews. Lastly, I would like to thank rest of the cybersecurity team for their active support and guidance.

Oulu, 25.03.2016

Tiandu Zhao

ABBREVIATIONS

AE	Authenticated Encryption
AES	Advanced Encryption Standard
BM	Base Measure
CBC	Cipher Block Chaining
CIA	Confidentiality, Integrity, and Availability
CIS	Center for Internet Security
CSS	Cascading Style Sheets
CVE	Common Vulnerabilities and Exposures
DM	Derived Measure
FIFO	First In, First Out
GCM	Galois/Counter Mode
H2	Hypersonic 2
HSIS	Host-based Security Instrumentation System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IP	Internet Protocol
IV	Initialization Vector
ISO	International Organization for Standardization
ISMO	Information Security Measuring Ontology
JSON	JavaScript Object Notation
MAC	Message Authentication Code
MVS	Metric Visualisation Software
NASL	Nessus Attack Scripting Language
NFR	Non-Functional Requirements
NIST	National Institute of Standards and Technology
OODA	Observe, Orient, Decide, and Act
REST	Representational State Transfer
SA	Situational Awareness
SDLC	System Development Life Cycle
SIEM	Security Information and Event Management
SHA	Secure Hash Algorithm
SMO	Security Measurement Ontology
SQL	Structured Query Language
SUI	System Under Investigation
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier

1. INTRODUCTION

In the modern world, more and more sensitive information is stored in computers, and the information has to be accessible all over the world. Cyberattacks have long been a nuisance for organisations, but as confidential information and critical infrastructure move into the digital world, the attacks may cause damage on a much larger scale. Compared to traditional attacks, cyberattacks are not location-based or physical, and they are extremely complex and fast [1]. For example, we have seen data breaches such as the Ashley Madison breach [2] and the PlayStation Network outage [3], where the personal information of millions of users was compromised.

New devices are connected to the Internet with increasing speed. Some of the newly connected devices may introduce vulnerabilities that compromise the security of the whole network. Proper security design and execution of security countermeasures – based on correct situational awareness (SA) – is able to prevent cyberattacks and mitigate security risks to networks and devices.

Organisations operate in different environments, and thus they prioritise security objectives and needs differently [4]. A one-size-fits-all approach is never going to be optimal as some of the systems will be over-protected and others under-protected [5] [6]. The security measurements should be adapted to the current environment. Measuring security provides a way to verify that the security controls are properly implemented. Moreover, security measurement is one way to build SA. Gaining SA is crucial for proper decision-making. Security-measuring instruments gather security data from systems and networks. However, this requires that proper security-measuring instruments can be built for data-gathering. The gathered data is analysed and visualised to build SA for administrators and decision-makers. A myriad of different security software solutions gathers security data to obtain an overview of the current security situation in the system. However, the solutions support custom measurements poorly and adaptive security measurement is lacking.

The lack of a framework and commonly accepted data models for system security instrumentation has led to a situation where the reusability and maintainability of the instruments are questionable. Similarly, building new security-measuring instruments produces ad-hoc implementations, instead of well-defined and reusable instruments. In addition, there are no generally accepted output formats for the measuring instruments. This work concentrates on these issues by identifying reusable and measurement-specific functions in security instrumentation, and proposing a framework and data model for adaptive system security instrumentation.

The Host-based Security Instrumentation System (HSIS) is the implementation of the framework; it supports creating and remotely managing adaptive security-measuring instruments on Windows and Linux systems. The HSIS enables security-measuring instruments to be reused, and it optimises the process of conducting security measurements. Moreover, the gathered data is securely transmitted from the observed system to the analysing entity.

The aim of the thesis is to enhance the process of conducting new adaptive security measurements on a host system, and to support the user in securely managing multiple instruments across the network. Hence, the contribution facilitates system administrators' and security auditors' work when security measures are applied.

2. BACKGROUND OF SECURITY MEASUREMENT AND TERMINOLOGY

This chapter presents relevant terminology and previous research. First, an overview of cybersecurity is presented, followed by a brief introduction to security engineering. In subchapter 2.3, security measurement and measures are defined and their properties are discussed. In subchapter 2.4, situational awareness in cybersecurity is explained. Finally, similar solutions are presented.

2.1. Cybersecurity

Confidentiality, *integrity*, and *availability* (the CIA triad) are considered the fundamental principles in information security [7]. *Confidentiality* ensures that unauthorised users are unable to read the content. This can be achieved by encrypting a message or by storing information in a secured location. *Integrity* means that modifications are detected or that an unauthorised user does not make modifications to the original content. In other words, the content has not been tampered with. Finally, *availability* ensures that authorised users have access to the content when needed.

ISO/IEC 27032 [8] defines cybersecurity as “*preservation of confidentiality, integrity and availability of information in cyberspace.*” Cyberspace can be seen as an environment that does not exist in a physical form and that is a result of different interactions between users, software, and systems and network devices that are interconnected [8].

It can be considered that, by definition, cybersecurity is part of information security that only happens in cyberspace. In addition, information security and cybersecurity may also involve other properties, such as authenticity, accountability, non-repudiation, and reliability [8]. However, Von Solms and Van Niekerk [9] argue that cybersecurity and information security are two separate concepts that intersect with each other. Cyberattacks on critical infrastructure such as power plants and power grids may cause the loss of integrity or availability of information, but the main impact will be on society and other infrastructures that are dependent on the electricity supply. The computer networks often support other organisational purposes [1].

Cybersecurity is not only about protecting the information in cyberspace, but also the users, devices, and infrastructures that can be affected by cyberspace. This includes attacks that use cyberspace as an attack vector to cause damage and disturbance outside cyberspace.

In this thesis, the term security is used to refer to cybersecurity, meaning security aspects of cyberspace [10].

Security can be modelled using the *objectives* that are set by the organisation to achieve specific security results [7]. The security objectives ensure that the confidentiality, integrity, and availability of *assets* are preserved. Figure 1 shows the relation of the security concepts. Security *risk* is defined as the uncertainty of not achieving security objectives (i.e., confidentiality, integrity, and availability) [7]. The *risks* are affected by the *security controls* that mitigate the *risk* [7]. *Security controls* consist of actions or processes to reduce risks. Thus, organisations may implement different technologies and policies designed to prevent, detect, and respond to risks.

Security controls consist of different *security mechanisms*, which can be parameterised. Security mechanisms support security objectives and protect assets [11]. *Threats* exploit *vulnerabilities* found in *security control* and *assets* [7].

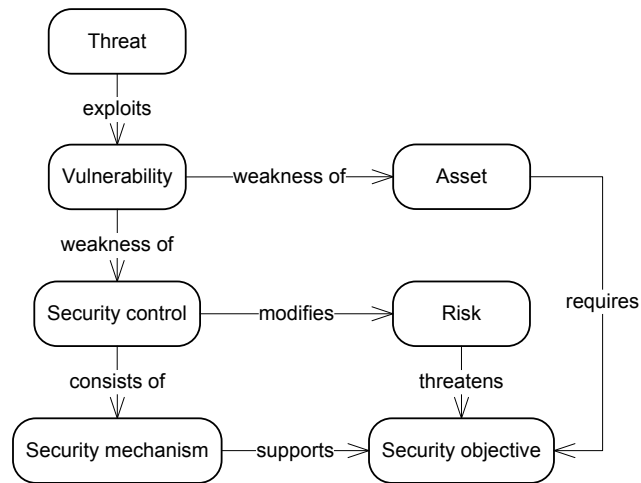


Figure 1. Security concepts.

2.2. Security engineering

Security engineering consists of building systems that are reliable during malicious use and attacks. Moreover, security engineering is also about using different tools, methods, and processes to design, implement, and test existing systems against cyber threats. [12]

Figure 2 visualises the system development life-cycle (SDLC). Early identification of security issues and mitigation of vulnerabilities lowers the costs of implementing security mechanisms to mitigate risks [13].

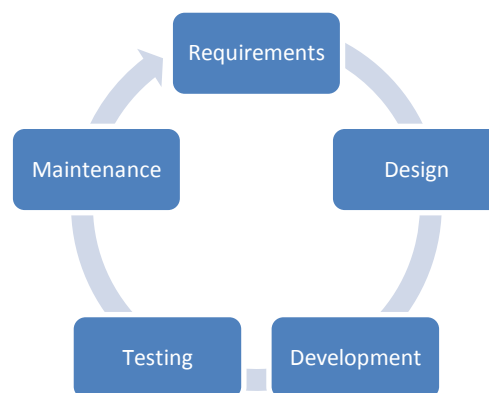


Figure 2. The phases of the system development life-cycle.

In the SDLC, the first step is to identify security requirements for the system. Systems have critical assurance requirements to prevent failures that would have catastrophic effects on users, customers, and even the environment that surrounds the system, such as in the case of a failure in a nuclear power plant [12].

In the design phase, architectural and design requirements for the system are established. Moreover, possible threats are modelled and an analysis of the attack surface is performed.

In the development phase of the SDLC, the system is implemented using reliable tools. In addition, one should avoid using functions, packages, modules, and libraries that are deprecated, to decrease the risk of using unsecure methods.

The system is tested and verified in the fourth phase. Testing includes performing a dynamic analysis that monitors the behaviour of the application for possible memory leaks, privilege issues, and other critical security issues. Moreover, the system can be fuzz-tested to induce failure in the system, revealing potential security issues that do not occur during normal operation.

The maintenance phase is the last step of the SDLC. The software is released to the public and its operation is actively monitored. Issues that have been detected during this phase, such as security weaknesses, are addressed in the next iteration of the SDLC, meaning that the issues become requirements for the next cycle.

The presented framework and the HSIS focus on the last phase. Attributes such as the basic information of the system (e.g. version number and status), messages or errors generated by the system (e.g. log files), and misconfigurations can all be observed from the system by applying security-measuring instruments. The attributes can provide valuable security information, but at the same time, the attributes can indicate issues that have not been detected during the previous phases of the SDLC.

2.3. Security measurements

As described above, organisations have different security objectives that are threatened by risks. Moreover, the risks are mitigated by security controls that consist of different mechanisms. The existence of these mechanisms, or the level of provided security, can be measured from the *System Under Investigation* (SUI).

The saying “you cannot manage what you cannot measure” also applies to security. Inevitably, there are always threats; however, it can be manageable by being aware of current situation, meaning by understanding which security mechanisms are operating and which are not. Nevertheless, it is possible to find the strongest and the weakest systems in an organisation by measuring different aspects of security from the systems [5]. Measurements can reveal potential security issues before the systems are compromised [5]. Therefore, resources can be allocated more efficiently to the systems that need them the most [14]. In other words, it is not reasonable to protect all assets at the highest level [15]. More importantly, the systems that are in danger can be systematically hardened after possible security issues have been identified [5]. Finally, security can also be measured for different assurance and compliance reasons, such as laws and ISO certificates [16].

Security measurements can be used to ensure that the organisation is heading in the right direction and that investments in security have been financially beneficial [5]. Measurements can also be used to answer different security-related decisions and questions [16].

However, measuring security may also induce unwanted consequences. Increased security may promote riskier behaviour in system users [16]. Moreover, measurements may also reveal information that can potentially be useful for the attacker [16]. Security measurements are capable of modelling only part of the SUI [14], which may not be sufficient to indicate the overall security level of the SUI [15]. In addition, two systems can be secure independently but not secure when they are connected [15].

Despite criticisms of security measurements, Watts pointed out that “just because outcomes are not predictable does not mean that there’s nothing to be done” [17]. Furthermore, Savola [18] concludes that even though security measurements have their shortcomings, they can be used to increase the security level and performance in systems. Finally, security measurements are the means of providing evidence of the systems’ security level [14].

Security rarely depends on one single attribute, but often on many smaller elements [11]. These elements can be further decomposed from the security objective all the way down to the base measures, meaning the decomposition is continued until a single measurable attribute that does not depend on other measures is found [4]. There are multiple strategies for the decomposition process [19]. However, anticipating all possible interactions and threats is close to impossible [16] due to the complexity of the environment caused by the vast amount of interconnected devices and systems. Furthermore, security consists of different security objectives that vary between systems and organisations [15].

Such decomposition may also oversimplify the situation and cause valuable information to be lost in the process [20] [21]. Even so, a good estimate of the security can be achieved when the SUI is small and simple [15]. Moreover, Savola [14] states that an adequate collection of security measurements can measure even complex situations.

2.3.1. Security measurement terminology

Security and security metrics terminology is not uniform, so it is good to know that security *metrics*, security *measurement*, security *indicators*, and even security *strength* are often used to describe the same concept [14]. The most widely used term is security *metric*; however, the term is misleading as security as a concept cannot be measured due to the complex properties of systems and attackers [22].

Metrics are defined as a description of data that is derived from measurements used in decision-making [11] [23]. Moreover, the *metrics* are defined as the distance between two values [24]. On the other hand, *measurements* can be defined as a set of operations with the goal of providing a single-point-in-time value of a specific attribute [23] [24].

The term *measurement* is more suitable in the context of conducting security measurements and obtaining measurement values. Therefore, the thesis will use the measurement terminology defined in Information Security Measuring Ontology (ISMO) [25], which is based on the Security Measurement Ontology (SMO) [24] and the ISO/IEC 27000 standard [7].

In the ISMO, base measures (BM) do not depend on other measures and they measure an *attribute* of an *object*. The BM is the foundation for derived measures (DM) and indicators [11]. DMs are derived from other BMs or DMs using a *measurement function*. A measurement function is an algorithm that combines two or

more BMs or DMs. Indicators are measures that depend on other measures and they are obtained by using an *analysis model*. The analysis model is associated with *decision criteria* that determine the value of the indicator. The interpretation of one or more indicators is called *measurement results*. Figure 3 illustrates the relation of the security measures.

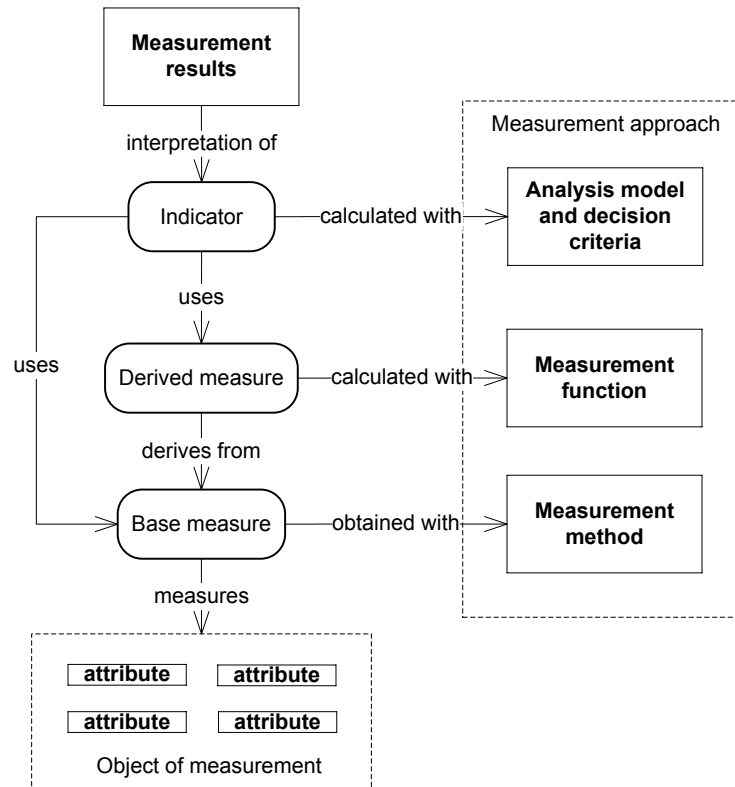


Figure 3. Measurement terminology.

Each of the measures has a *measurement approach* to obtain the measurement value [25]. The code implementation of the measurement approach that is capable of obtaining the value is called the *security-measuring instrument* or simply *instrument*. The main contribution of this thesis is the development of these reusable instruments. Table 1 shows an example of each measure in the process of determining the number of strong passwords in an organisation. Here, the object of measurement is the database and the attributes are passwords in the database. The first base measure (BM #1) determines the total number of passwords stored in the database. The second base measure (BM #2) determines the number of passwords for each user that are in line with the organisation's policies. DM calculates the total number of passwords that satisfy the requirements by summing the values of BM #2. Lastly, the indicator indicates the ratio of passwords that satisfy the requirements. The analytical model determines the final value for the indicator. In this example, the indicator yielded the value 'DECENT'. [26]

Indicators are used to satisfy an information need [7]. The information need defines the necessary observation to manage objectives, risks, and issues. Due to the scope of the thesis, the indicators are used to indicate the level of security mechanisms, such as the strength (indicator) of user authentication (security

mechanism). Figure 4 shows the big picture of the security concepts with related measurements.

Table 1. Example of base and derived measures, and indicator.

	BM #1	BM #2	DM	Indicator
Name	Number of passwords.	Number of passwords that satisfy the requirements for each user.	Total number of passwords that satisfy the requirements.	The ratio of passwords that meet the requirements.
Measurement method	Count the number of passwords in the database.	Ask the users if they follow the guidelines for passwords.	-	-
Measurement function	-	-	\sum of BM #2	-
Analytical model with decision criteria	-	-	-	IF BM #1 / DM > 0.9: GOOD IF BM #1 / DM < 0.5: BAD ELSE: DECENT
Value	1239	-	929	DECENT

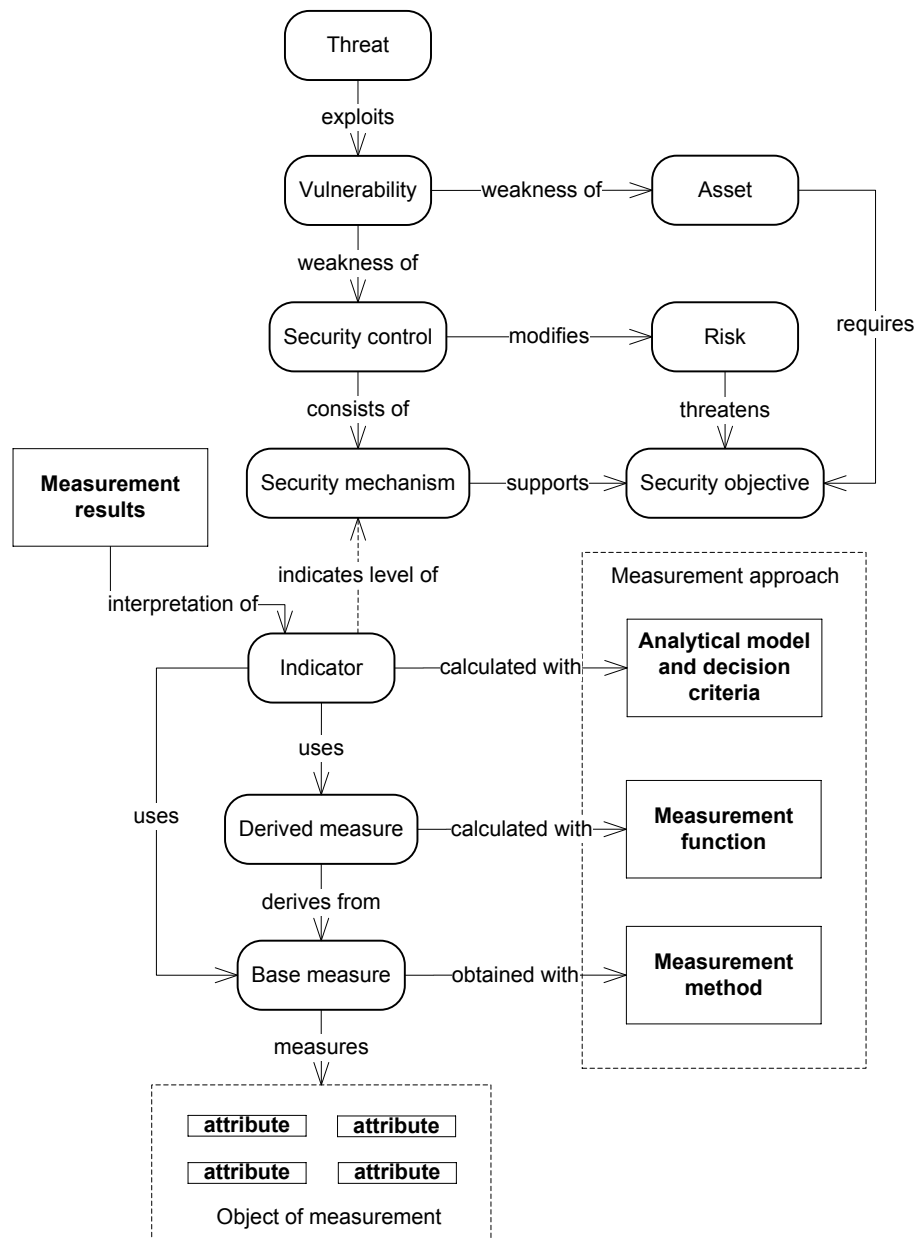


Figure 4. Security concepts with measurements.

2.3.2. Properties of security measures

Savola [18] collected eighteen properties that a good security measure should have. Out of all the good properties, it was found that security experts consider correctness, measurability, and meaningfulness as the most important criteria for a security measure [22]. The security measures should also be usable [22], context specific [27], and SMART [23], meaning specific, measurable, attainable, repeatable, and time-dependent.

Often the security level of a system cannot be evaluated on a scale from one to ten or from 'bad' to 'good'. Thus, the focus has mainly been on whether the change or

existence of an attribute makes a system ‘less bad’ than before [6] [16]. For instance, the Center for Internet Security (CIS) [28] and MITRE’s Common Vulnerabilities and Exposures (CVE) [29] have a comprehensive list of security benchmarks and vulnerabilities for different systems and software. Reducing the number of known vulnerabilities that exist in a system does not make the system secure, but rather less vulnerable than before [6].

Security measurements can be divided into two different types: qualitative and quantitative. Quantitative ones are countable, such as the number of users that have administrator rights. On the other hand, qualitative measurements may not be countable or are represented with a category value such as 1, 3, and 5, or “low”, “medium”, and “high”. Quantitative measurements often require interpretation.

There are some criticisms of qualitative measurements as they heavily rely on subjective observation. Some even consider qualitative measurements to be ratings [27] rather than measurements. However, taking a stand on whether qualitative measurements should be accepted as a valid measurement or not, is beyond the scope of the thesis. Hence, the HSIS is designed to process both quantitative and qualitative measurements.

2.4. Situational awareness

Understanding the current situation is one of the first steps in cyber defence [30]. Gaining SA is important in supporting decision-making on which security actions should be taken, which means being aware of which systems are at risk. The process of reacting to situations is often formalised using the OODA loop [1] [31]. The loop consists of four phases: *Observe*, *Orient*, *Decide*, and *Act*.

In the cybersecurity defence context, the observe phase consists of gathering data from devices or network, that is, instrumenting the SUI. In the orient phase, the current situation is understood and possible weaknesses are detected. Next, in the decide phase, the security operator decides which actions are to be taken to overcome the weaknesses. Lastly, the countermeasures are implemented in the SUI.

Observing the environment is possible by instrumenting the SUI. The instruments are software-based solutions and they measure different aspects of the SUI, meaning that the instruments are the security-measuring entities. Interesting information may also be secondary information, that is, generated by other software. For instance, software log files are secondary information.

However, the SUI often operates in a dynamic environment where the environment and the need for different measurements change. In addition, the relevance of the measurements varies accordingly. For instance, monitoring for malicious network packets from a closed port is redundant. Building SA relies heavily on observation, and so data gathering should always respond to changes in the environment.

Klein et al. [31] propose the following related activities in the OODA loop:

- 1) Deployment of reliable security-measuring instruments
- 2) Secure transmission of the data to an analysing entity
- 3) Analysis, correlation, and visual display of the data and situation

- 4) Decision support for countermeasures
- 5) Specification and parameterisation of responses
- 6) Reliable deployment of countermeasures

The activities are aligned with the OODA loop in a holistic fashion, so that one activity may not exist without the others [31]. Figure 5 visualises the alignment with the loop. Reliable instruments are needed to gather data from the SUI. Gathering data (Activity 1) is the starting point of gaining SA, followed by securely sending it to an analysing entity (Activity 2), where the data is analysed, correlated, and visualised (Activity 3) along with other data gathered by other security-measuring instruments. This further helps the administrator in building SA. Analysing entities may aid the administrator in choosing suitable countermeasures to deal with the situation (Activity 4). Lastly, based on the gathered data and analysis, countermeasures are parameterised (Activity 5) and implemented in the system (Activity 6).

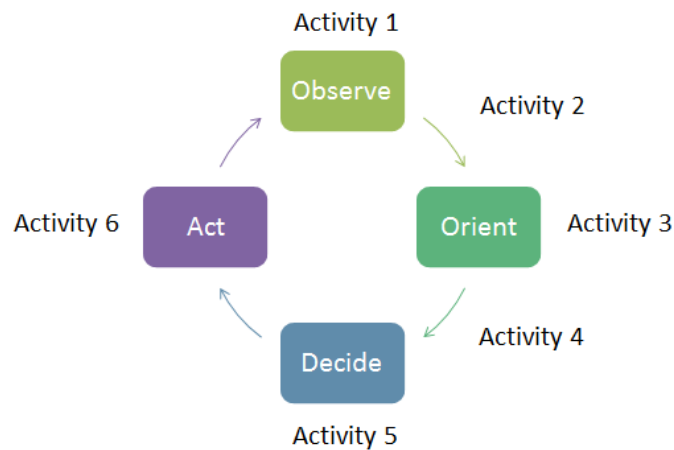


Figure 5. The related activities aligned with the OODA loop.

A myriad of different software solutions that produce information can be used to evaluate the security level. The solutions include packet analysers such as the Wireshark [32], port scanners like nmap [33], auditing tools such as Lynis [34], and many consumer-level software solutions such as firewall and antivirus software. Some of the solutions may have integrated reactive mechanisms to prevent an issue from increasing. For instance, firewalls may block malicious packages, and vulnerability scanners may have mechanisms to fix the found vulnerabilities.

However, cybersecurity defence is often concentrated on the detective or on the reactive mechanisms, but rarely on what happens between the two stages [31]. The thesis will drill down the gap between the detective and the reactive by presenting the HSIS, which focuses on the two first activities (Activities 1 and 2).

2.5. Similar solutions

Multiple solutions are capable of gathering security information from computer systems. The solutions use two types of approach to measure security from a host

system: ‘agent’ and ‘agentless’ measurement. In the agent-based approach, an agent operates inside the SUI and sends the gathered data to the server, where it is often processed and analysed. In the agentless approach, the server measures the SUI by remotely accessing the system using pre-entered host credentials or by querying the SUI for available data. For instance, the name and version of a web application are often available without accessing the system. In addition, commercially available solutions are often bundled together with rich features, such as automatic reporting and data visualisation.

The functioning of the security-measuring *instruments* is similar to that of the agent-based solutions, and thus, some of the solutions that offer agent-based measurement are presented.

For instance, Nessus [52], Snare [53], and Assuria Cybersense [54] all offer an agent-based measurement. The selected solutions offer continuous measurement and monitoring of the SUI by introducing agents to the SUI. The agents collect a large amount of data, including vulnerability, compliance, and system information. Significant benefits of using host-based scanning over agentless scanning include the performance gain and the capability of performing offline scanning, which is when the device is not connected to the local network. In addition, there is no need to save the host credentials in the server with the agents.

The presented solutions’ agents are capable of measuring a comprehensive number of attributes from the SUI. In addition, users can choose appropriate measurements, and thus, users can customise the agents for each SUI. Not all solutions support user-created measurements, but Nessus, for instance, supports *plugins* that allow the Nessus agents to check for a given flaw in the SUI. The user can create custom plugins using Nessus Attack Scripting Language (NASL).

However, the agents are not capable of adapting to environmental changes, and thus, data gathering is not optimal in all situations. Moreover, adding custom measurements is poorly supported by the solutions.

This thesis defines the common and non-common functions of security instrumentation. In addition, reusable and *adaptive* security-measuring instruments are presented. Finally, the HSIS supports user-created measurements that are written in a more general programming language.

3. FRAMEWORK FOR ADAPTIVE SECURITY MEASUREMENT

This thesis presents a framework that aims to assist and optimise the process of creating adaptive security-measuring instruments. The HSIS is built upon the presented framework. Moreover, in this chapter, the common components found in the instruments are presented.

In the first subchapter, an overview of the intended instrumentation is presented. In the second subchapter, requirements for the framework are defined. Moreover, specifications that apply to all instruments are also specified. Lastly, a conceptual architecture that fulfils the requirements and the specifications is presented.

3.1. Overview of system instrumentation

A vast amount of data is available that support the process of measuring the security level. However, the absence of interfaces and the incompatible formats between the data-generating software and the analysing entity have led to a situation where the information is accessible only locally or not at all.

Although there are a multitude of aspects that can be measured in a system, from the security-measuring instrument's perspective, they all share similar functions and requirements [36]. The main functions are gathering data from the system, processing and formatting the gathered data, and sending it to an analysing entity for post-processing. These functions can be further decomposed into common and non-common ones; common functions are the same for all measurements and non-common functions are unique to each measurement.

Security-measuring instruments gather data from the SUI for security measures. The instruments are defined as the most basic unit that performs measurements on the SUI and sends the results to an analysing entity. In other words, *instruments* implement the *measurement approach*, as defined in the previous chapter.

The framework for security-measuring instruments supports instrumentation on the SUI. Figure 6 shows an overview of an example network in an organisation, consisting of Windows and Linux workstations and a database server. Each system can have multiple instruments (cyan circle). However, the gathered data is often very detailed and contains many data bits that may require human interpretation to support the observation. Therefore, the data is sent to an analysing entity. For instance, Metric Visualisation Software [49] (MVS) or Security Information and Event Management (SIEM) solutions can be used to process and visualise the data gathered by the instruments.

The instruments consist of a component *probe* (yellow circle), whose only function is to obtain the measurement value from the SUI, meaning that the *probe* is the non-common component of the security-measuring instrument. The *probe* is the component in the instrument that is capable of extracting the data from the SUI, which is the *measurement approach* of BM, DM, and indicator. All other functions, such as message formatting, are included in the common components of the instrument. Thus, the operations of the probes are independent from other components of the instrument, and the separation allows probes to be created separately from the instrument.

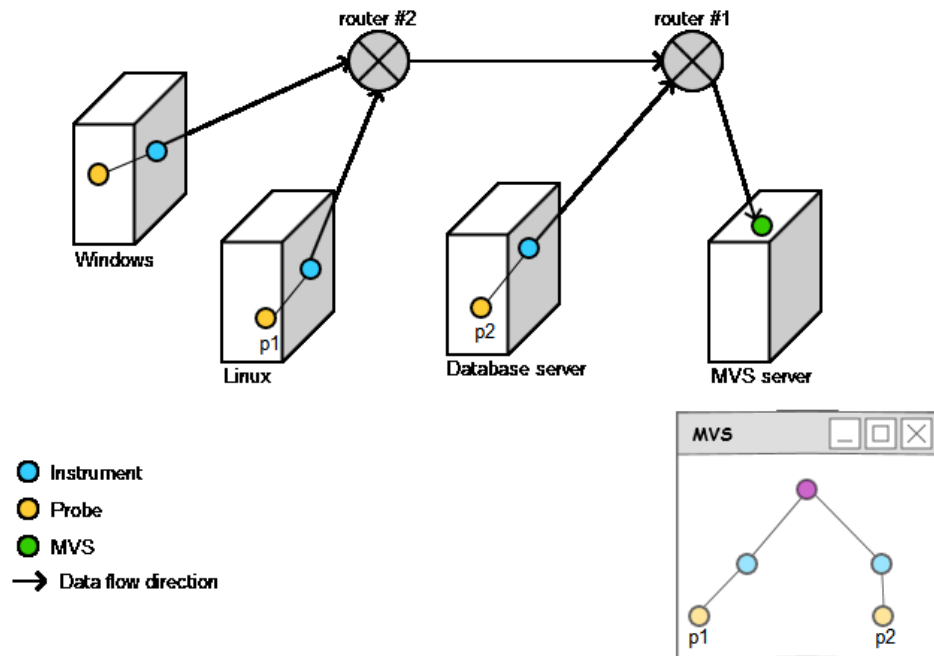


Figure 6. An example of instrumentation on three systems in a network.

Data gathering, processing, and visualisation can be classified into four different architectural layers. Figure 7 visualises these four layers. In the architecture, each layer can control and command only the layer directly below it. For instance, Layer 2 may command and control only Layer 1. On the other hand, data flows in the opposite direction, so that only Layer 1 has access to the data in Layer 0, and only Layer 2 has access to Layer 1, and so on.

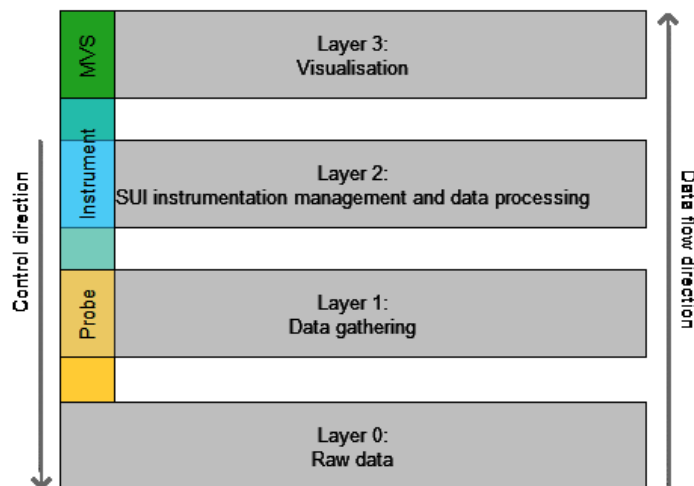


Figure 7. Architectural layers of the instrumentation process.

The first layer, Layer 0, consists of the unprocessed data (data source) from the probe's perspective. The data can originate from the observed system itself or from other software, such as auditing tools, firewalls, and antivirus software. The data exists in many formats and types, including log files, scan results, and policy settings. The probes (Layer 1) gather the data located in Layer 0. In the next layer, the instrument processes the gathered data and prepares it for transmission (Layer 2). In addition, the instruments have different settings that can be modified (Layer 2), such as the IP address for the analysing entity, and the measurement frequency. Measurements are analysed at the top of the architectural layer (Layer 3). The user interaction happens at the very top of the layers; only the visualisation is visible to the user.

3.2. Requirements for the framework

There are three different actors in the instrumentation process: probe designer, instrument manager, and visualisation software user. Figure 8 shows the use cases of security instrumentation. Probe designers should be able to create new probes that are compatible with the instruments. They also ensure that the probe is functioning properly, so that data is gathered according to the specifications. Instrument managers are responsible for deploying and managing the security-measuring instruments. They should be able to deploy new instruments and remove unnecessary instruments, and change settings of existing instruments in the SUI. Lastly, the visualisation software (e.g. MVS) user should be able to see the data sent by the instruments.

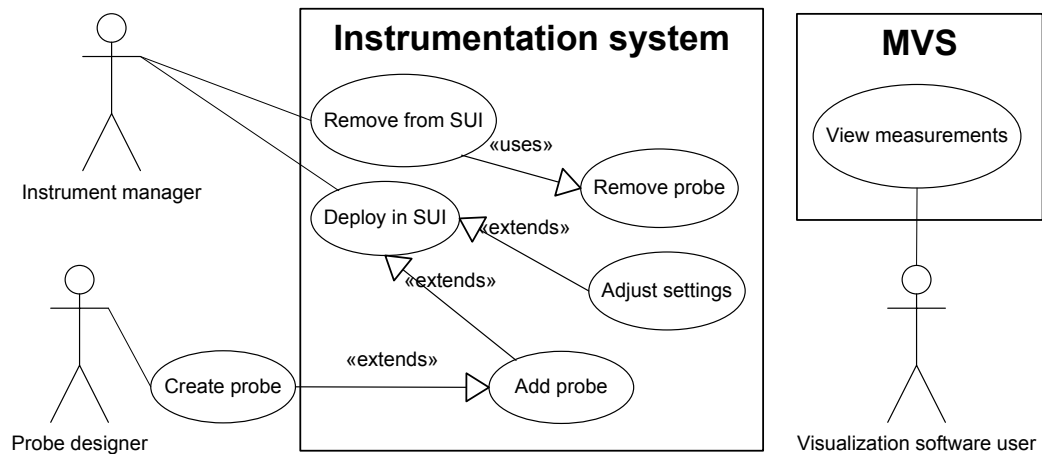


Figure 8. Use cases for the security-measuring instrument and the MVS.

Requirements for the security-measuring instrument, and for the probes, are strict as decisions, and the evaluation of security is based on the measurements provided by the instruments. For instance, an error in the BM may change the indicator value. Consequently, the indicator might indicate a higher level of security mechanism, which may in turn suggest that the risk is low even though it is at critical levels. The closer the error is to the actual measurement, the wider the effects it has on the whole security-measurement chain.

Requirements for the framework are split into two categories: functional and non-functional requirements (NFR), listed in Table 2 and Table 3, respectively. The NFR are based on the framework presented by Kanstrén et al. [36].

Table 2. Functional requirements for the framework.

ID	Description
R01	Gather security information.
R02	Gather in real time or at a given frequency of measurement.
R03	Format the output message in a uniform format.
R04	Send gathered data to an analysing entity.
R05	Run-time adaptation (capability to react to changes around the instrument).
R06	Remotely manageable and configurable

Functional requirements R01 and R02 are related to the definition of system instrumentation. Each of the probes, meaning the component in each instrument that is responsible for the data-gathering, has to be capable of gathering security information from the SUI at a given frequency of measurement. Requirements R03 and R04 define the operations that support data-gathering. It is crucial for gaining SA that the data gathered by the instruments are sent to an analysing entity. In this work, the results are forwarded to the MVS, where they are visualised and analysed. In addition, the instrument should be capable of adapting (R05) to changes in the environment and measurement needs by optimising its functional processes. Finally, the instruments should be remotely manageable and configurable (R06).

NFRs R07-R16 define quality aspects of the framework. Each instrument has a set of attributes (R07) that is used to describe the instrument and the obtained measurement. These attributes are the basis for the output message format. Instruments are deployed in many types of environment and system; therefore, the functional components in the instruments should be modular (R08). Modularity enables users to customise the instruments to satisfy the requirement needs in a much wider environmental scope than is achievable with the run-time adaptation.

On the other hand, intrusiveness (R09) needs to be minimised, as probing always affects the performance of the observed system to some extent (the so-called probe effect). In addition, isolation from the SUI (R10) ensures that failure in the instrument is not propagated to the SUI itself. Security-measuring instruments should be highly dependable (R11) and continue to gather data as long as possible, even in the case of system failure. This ensures that the problem can be analysed afterwards.

In general, probes tend to gather sensitive information from the system, and therefore, it is critical to protect the information (R12) from unauthorised users. Confidentiality can be ensured by encrypting the gathered data, using a secured connection, and implementing authentication. Necessary countermeasures should be taken to protect the information. Despite all the countermeasures, at the same time information should be available to authorised users when needed (R13). The instrument should operate on all major operating systems (R14). Moreover, the

instrument should not be tied to a specific system (R15), so that it is reusable in many operating system environments. Finally, the instruments should support user-created probes (R16).

Table 3. Non-functional requirements for the framework.

Name	ID	Description
Common attributes	R07	Each instrument has mandatory fields that are used to identify and describe the instrument and the measurement approach, that is, the probe.
Modularity	R08	Instruments should consist of separated components that can be easily exchanged and customised.
Non-intrusive	R09	Deployed instrument should minimise the performance effect on the SUI.
System isolation	R10	Failure in instruments should not propagate to the SUI.
Highly dependable	R11	Instruments should keep gathering data in all situations, to allow the administrator to review the situation with the most accurate and recent data (e.g. cause of crash or system failure).
Security	R12	Instruments should be protected from possible malicious usage and the results should be reliably sent to an analysing entity.
Availability	R13	The data gathered by the instruments should be accessed within a reasonable time.
OS independent	R14	The operation of the instruments should not depend on the operating system.
Reusable	R15	Instruments are not tied to a specific system, meaning that the instruments can be reused in other systems.
Custom measurements	R16	Instruments should be compatible with user-created probes.

3.3. Conceptual architecture

A conceptual architecture is proposed to satisfy the requirements defined above. Thus, additional elements are added compared to the setup shown in Figure 6.

To eliminate the common parts required to instrument the system, in the conceptual architecture the security-measuring instruments may have multiple *probes*. This not only reduces the overlapping functions of multiple instruments in the SUI, but also facilitates instrument management. In addition, a new subsystem

called *Manager* is introduced to remotely manage and configure the instruments (R06). All the instruments are accessible from Manager. Figure 9 shows the relation of the instruments and Manager. *Manager* (red circle) is added as part of the system and acts as a conjunction point for the information sent by the instruments. Analysing entities may get the measurements straight from the instruments or alternatively via Manager.

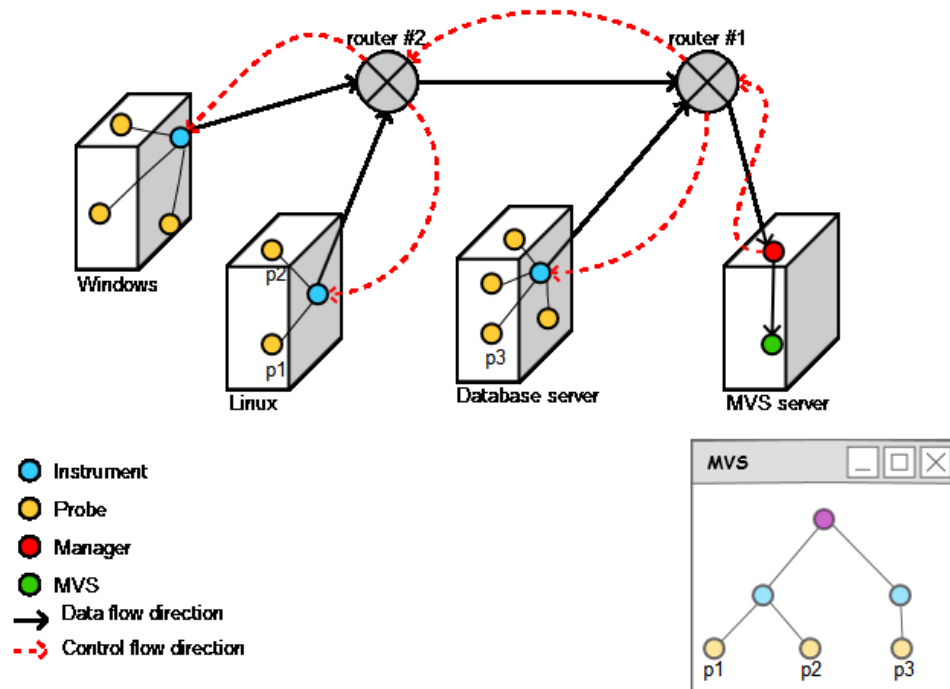


Figure 9. Conceptual architecture of instrumentation in a network with all the required subsystems.

The framework of two different subsystems: a security-measuring *instrument* and *Manager*. The subsystems operate independently and do not rely on each other. However, Manager centralises instrument management and stores measurements and other instrument-related information in the database for offline access. This reduces the impact on the performance in the SUI by limiting the amount of information stored and processed by the instruments. Each instrument is controlled and managed by only one Manager. The relation of the subsystems is shown in the component diagram in Figure 10.

Manager should be capable of changing the operational aspects of the instrument subsystem. This includes aspects such as changing the rules that guide instrument adaptation, the measurement frequencies, and how often the measurement values are updated to Manager. Moreover, the common attributes of the instrument and the probes can be remotely modified. Lastly, the measurements are sent to the MVS, or any other third-party solution, using an appropriate interface (R04).

The instrument subsystem consists of following components (R08): *controller*, *forwarder*, and *probes*. Table 4 describes the components in more detail. In the framework, one instrument may have multiple *probes* that measure different aspects of the SUI. This kind of structure eliminates the overlapping functions as opposed to having multiple instruments with only one probe. The *controller* component controls

the whole instrument subsystem. Its main responsibility is to configure how the instrument operates, for example how often the results are sent to Manager, and how the probes are operated. The last component in the instrument is the *forwarder*. Its main tasks are to construct the output message in a unifying format (R03) by including the mandatory attributes, and to send securely the message to Manager or directly to an analysing entity.

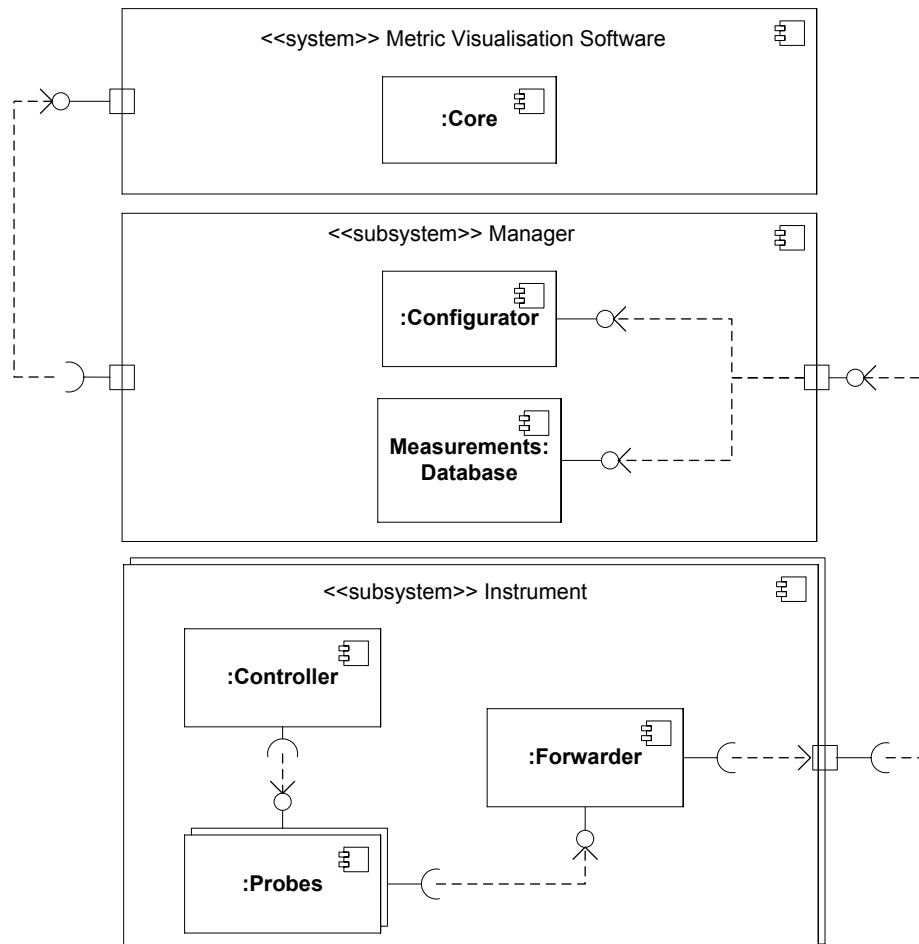


Figure 10. Component diagram showing the relation between the instrument, Manager, and the MVS.

Requirement R12 defines that, for security, communication between the two subsystems should happen over a secure connection. This is achieved by encrypting all the messages using well-proven methods. Moreover, the integrity of the output messages is ensured, so that the messages have not been tampered with. One-way cryptographic hash functions can be used to ensure the integrity. In addition, each instrument *requires* a separate secure connection to communicate with Manager. This is to prevent malicious use by limiting the access to other instruments, so that one compromised instrument does not compromise other instruments.

As described in Table 3, requirement R07 defines that each instrument has attributes that are used to identify and describe the instrument and its probe. However, in the conceptual architecture, the instrument may have more than one probe, and thus, it is necessary to describe the instrument and the probes separately.

Each instrument has a unique *identifier* (ID) and an optional name. Similarly, each probe has a set of attributes that are listed in Table 5. Each probe has five different fields to describe the probe and the measurement value.

Table 4. Description of the subcomponents in the security-measuring instrument.

Component name	Description
Controller	The controller is the main component of the instrument that ensures the instrument is operating with the correct parameters and configurations.
Forwarder	The forwarder constructs the output message that is securely sent to Manager or an analysing entity.
Probes	Probes conduct the measurements on the SUI according to its specification.

The first field is a unique ID for the probe. The ID distinguishes similar probes from each other. The *name* and *description* fields describe the probe function in plain English. This ensures that users understand the purpose of the probe. The *type* field describes the measurement value and guides other systems in parsing and analysing the measurement value. Lastly, each probe has a field for the *measurement value* itself and for a *timestamp* to pinpoint when the measurement was conducted.

Table 5. List of necessary attributes to describe probes.

Field name	Description
ID	Unique identifier.
Name	
Description	Short description of what the probe is supposed to measure.
Type	Indicates the type of measurement value, such as integer or Boolean.
Value	The measurement value obtained by the probe.
Timestamp	Indicates when the measurement was conducted.

In addition to the mandatory information, additional fields can be included. For instance, the status of the instrument or more detailed information on the probes can also be included in the output message that is sent to Manager.

The Manager subsystem is the main control unit of the whole framework, meaning that it controls and manages all the connected instruments. The subcomponents of Manager are described in Table 6. Manager receives messages from the instruments and stores them permanently in a *database*. This way, the

measurements are viewable even when the instruments are offline (R13). The measurement values, and other information, are accessible by the analysing entities via an interface. Alternatively, Manager may send the measurement values directly to the analysing entity. The other main component in the manager is the *configurator* unit. The unit facilitates remote control and management. The instruments are obliged to follow the *instructions* created by the *configurator*.

Table 6. Description of the subcomponents in Manager.

Component name	Description
Configurator	Manages the security-measuring instrument by commanding the instruments to operate under new parameters.
Database	Permanently stores all the content sent by the instruments.

Requirement R06 states that instruments are remotely configurable and manageable. The *instructions* contain parameters that change the behaviour of the instrument and the probes. They are also used to change the probes' and instruments' attributes. Table 7 describes the attributes that can be modified in the instrument. The run-time parameters for each probe can be modified using the *instructions*. Moreover, the *instructions* can also alter basic information of the instrument and the probes. The instruments run on their default settings unless they are modified by the *instructions*, thus, the *instructions* overwrite the default settings.

Table 7. Aspects of the security-measuring instrument that can be remotely modified.

Parameters and attributes	Description
Probe operation	Controls which probes are active and measuring and which are inactive and waiting for additional actions. Moreover, the measurement frequency of the probes can be adjusted.
Name for probes and instrument	Changes the names to be more descriptive and identifiable.
Description of probes	Changes the descriptions to be more accurate and detailed.

4. THE HOST-BASED SECURITY INSTRUMENTATION SYSTEM AND TESTING

This thesis presents the HSIS, which is based on the framework presented in the previous chapter. The chapter is arranged as follows. Firstly, the structure of the instrument and Manager are described. Secondly, the functions are described in detail. Lastly, the HSIS is tested in three test scenarios.

4.1. Concrete architecture

The concrete architecture of the HSIS follows the conceptual architecture displayed in Figure 9. Each SUI has one security-measuring instrument that can measure multiple attributes of the system using one or more probes, as described in subchapter 3.3. Python and Java programming languages are used to implement the instruments and Manager, respectively. Moreover, JavaScript Object Notation (JSON) is used to encode all the traffic between the subsystems. To satisfy security requirement R12, all JSON-encoded output messages go through authenticated encryption (AE).

The structure of the instrument and Manager are described in subchapters 4.1.1 and 4.1.2, respectively.

4.1.1. Structure of instrument

The first subsystem of the HSIS is the *instrument*. Figure 11 shows a detailed component diagram of the instrument and the folder where the probe files, or probes, are located. The probes are written in the same programming language as the instrument, which is Python.

The security-measuring instrument consists of several operational components: the *controller*, *forwarder*, *probe loader*, and *probe operator*. The design of the instrument follows design pattern Mediator [37], where the components do not directly communicate with each other and the components operate independently; see modularity requirement R08 in Table 3.

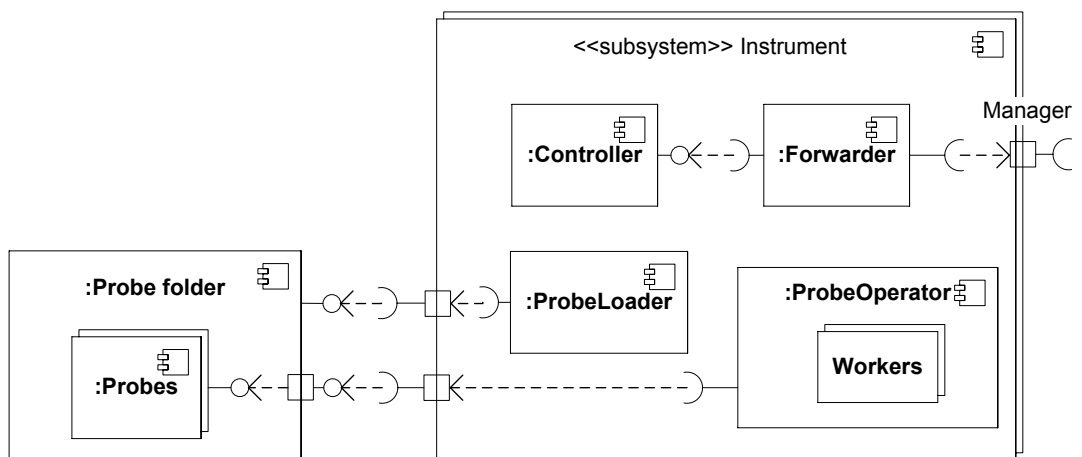


Figure 11. Detailed component diagram of the instrument.

The *controller* ensures that all settings and parameters are properly loaded and saved during run-time. The *forwarder* constructs the output messages and communicates with Manager. As the *instrument* dynamically handles the *probes*, the process of loading and operating the probes are split into two separate components. The *probe loader* component loads the probes and necessary information into the instrument. On the other hand, the *probe operator* is responsible for operating the probes, which means conducting the measurements based on the information loaded by the *probe loader*.

The first component of the instrument is the *controller*. It configures the instrument and the probes. The *forwarder* enables the communication with Manager and retrieves *instructions*, which are then passed to the *controller* for processing and deployment. In addition, the *forwarder* is also responsible for formatting and securing the messages that are sent to Manager.

The *forwarder* component may use normal or so-called ‘simple’ mode to format the messages. Simple mode reduces the load on the network by eliminating unnecessary information. Some fields in the message (the common attributes) do not change during two consequent connections to Manager and, therefore, the unchanged ones can be omitted from the message; only the fields that have changed are included in the message.

After the *forwarder* has successfully updated the measurements to Manager using HTTP library *requests* [38], Manager may return *instructions* in the response body. Figure 12 visualises the *instructions*, and four sections can be distinguished: *unload*, *load*, *probes*, and *instrument*. The *unload* section describes which probes have to be unloaded from the instrument. On the other hand, the *load* section contains a list of probes that have to be loaded by the instrument from Manager. A hash value is included to ensure the integrity of the loaded files. Lastly, the *probes* and *instrument* section allows Manager to configure settings for the loaded probes and the instrument, respectively. Any configuration in the instrument is automatically stored in an external configuration file by the *controller*.

```
{
  "unload": [{
    "uuid": "f5cac852-c92a-4174-832a-5a21a788852e"
  }],
  "load": [{
    "filename": "password_complexity.py",
    "hash": "2299538b49f79c4055b49f993c1ba0db084ce4b367fee61215132668d8620ba4"
  }],
  "probes": [{
    "uuid": "1f89ef6f-10c6-4f61-aa47-c62396e09863",
    "name": "A new name for the probe",
    "description": "A new description for the probe",
    "interval": 25,
    "dependency": "IF a_main_probe.py IS EQUAL TO TRUE",
    "mode": "2",
  }],
  "instrument": {
    "name": "A new name for the instrument"
  }
}
```

Figure 12. An example instruction constructed by Manager.

The *probe loader* monitors the probe folder, which contains the probes for possible changes. The component automatically loads probes from the folder into the instrument as an instance of Probe class, thus, the probes are not hard-coded in the instrument. The class contains the necessary information for the probe operator to operate. On the other hand, probes that are removed from the folder are also removed from the instrument, meaning that the probes can be loaded and unloaded during runtime by modifying the content of the folder.

After the probes have successfully loaded into the instrument, the *probe operator* evaluates the probes based on the information stored in the Probe class. The component creates a pool of workers with the task of operating the probes, that is, conducting the measurements. The probes are assigned to the workers once the *probe operator* is satisfied with the evaluation results. The workers return the measurement value once the probe has finished the measurement.

4.1.2. Structure of Manager

The other main subsystem is Manager, and it allows administrators to remotely manage and view instruments and probes. Moreover, the measurements from the probes are stored in a database and the measurement values can be automatically forwarded from Manager to the MVS. Manager consists of seven different classes. The relation of the classes is shown in Figure 13. The *core* is responsible for connecting the other classes together, and the *configurator*, *file handler*, and *database* interfaces access and process the instructions, database, and probe files, respectively. The *representational state transfer (REST)* interface provides the interface to manage and access Manager. *Access control* determines whether requests from the *REST* interface are authorised before they are forwarded to the other classes. Lastly, the *updater* updates the measurement values in the MVS (R04)

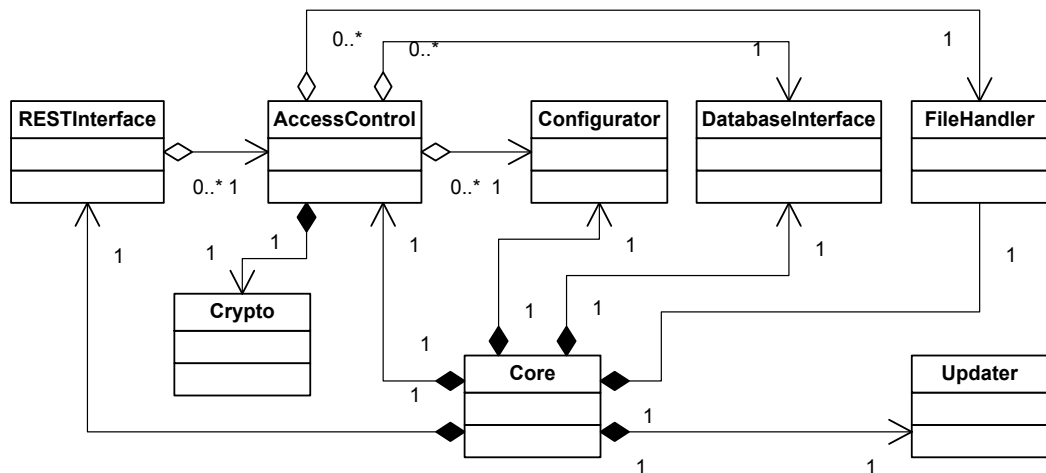


Figure 13. Simplified class diagram of Manager.

The *configurator* class constructs the *instructions* for each instrument (R06). The *instructions* are modified by the requests from the REST interface, and they are automatically returned to the instruments in the response body. The file handler handles the probe files that are downloaded by the instruments. Its main task is to

find the correct probe file from Manager and prepare it for instruments to download it.

The content of the message sent by the instruments is permanently stored in the Hypersonic 2 (H2) [39] database. All interaction with the database goes through the *database* interface. In addition, the interface also parses the messages, prepares and constructs the Structured Query Language (SQL) queries, and encodes the result set in JSON.

H2 is an embedded database that can be bundled together with Manager. The downsides of embedded databases include instability and the lack of some of the features that are offered by larger databases. However, H2 is compatible with the SQL grammar found in Microsoft SQL [40], MySQL [41], and Oracle [42]. Moreover, the H2 database outperforms other embedded database engines, such as Derby [35], in multiple cases [43], which improves Manager's performance in managing multiple instruments.

The *REST* interface is based on the HTTP protocol, and it is used to provide a way to manipulate and access Manager. It also enables the instruments to update their measurement values to Manager. The information stored in the database is accessible from the *REST* interface.

Lastly, the *access control* class verifies all requests from the *REST* interface before granting access to other classes. In addition, *access control* uses the *crypto* class to encrypt and decrypt the traffic between Manager and the instruments. Verified and authorised requests are passed to the *database* interface, *configurator*, or *file handler* for further processing. On the other hand, unverified or unauthorised requests are automatically rejected by access control. The process is visualised in Figure 14.

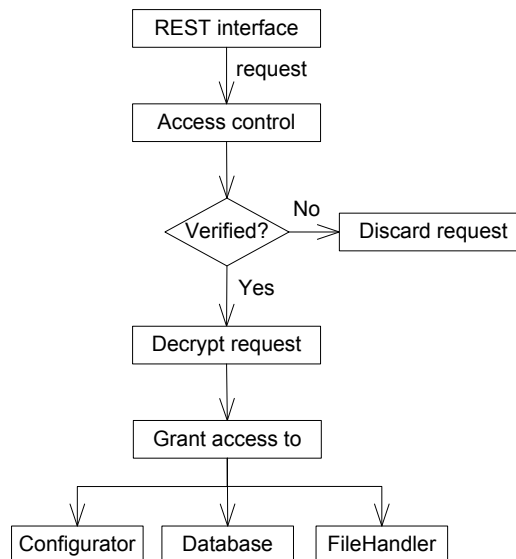


Figure 14. Flow chart of the verification process.

Dashboard is a web application that allows administrators to manage the HSIS remotely. The web application is accessible by entering the root uniform resource identifier (URI), such as <http://127.0.0.1>, in a browser or on a mobile device. The web application design enhances the remote manageability (requirement R06) of the

HSIS. Moreover, Dashboard demonstrates the REST interface provided by Manager. The Bootstrap [44] front-end framework is used to develop Dashboard. Bootstrap offers various design templates, based on HTML and Cascading Style Sheet (CSS), for web development. Moreover, Bootstrap is paired with the web application framework AngularJS [45]. Figure 15 visualises the user interface. Dashboard shows the information sent by the instruments in one single page. In addition, the available probes and files that can be downloaded by the instruments are listed at the bottom of the page.

🏠 Instruments Probes Available files Instrument key
15:55:39:958

Instruments

#	Name	IP address	OS	Timestamp	
0227334a-4fca-4ff7-8711-b16a75b6d076		127.0.0.1	Windows	2016-01-08 12:43:36.84	▼
0c1980c2-95d8-4ac1-b74c-b20a096a26d9		127.0.0.1	Windows	2016-01-15 13:19:35.937	▼

Probes

#	Name	Description	Type	Value	Timestamp	
e86404c3-92ef-462c-801d-d935c128d7ac	Antivirus software name	Retrieves the name of the AV that is installed in the host system	STR	F-Secure Client Security 11.60	2016-01-08 12:43:31.833	▼
d09ca76b-33ca-4b7d-ba4e-e98e58bbac24	Password age	Current user's password age in days	STR	71	2016-01-08 12:43:25.716	▼

Value	Timestamp
71	2016-01-08 12:43:25.716

Links
d973abae-e08e-4a51-a85f-acf5fbc62c43 ✕

Available files

File name	
5s_networkpackets.py	+
antivirus_name.py	+
firewall_status.py	+
nmap_script.bat	+
open_ports.py	+
password_age.py	+
password_complexity.py	+
timezone.py	+

```
import subprocess

name = "Antivirus software name"
description = "Retrieves the name of the AV that is installed in the h
interval = 5

def run():
    pout = subprocess.Popen(
        'WMIC /Node:localhost /Namespace:\\\\root\SecurityCenter2 Path
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        shell=True).communicate()[0].strip()
    return pout.decode('utf-8').split("=")[1]
```

Figure 15. Dashboard for Manager.

The information for instruments and probes is split into separate tables. The Probes table can be filtered to show only the relevant probes by selecting an instrument from the Instrument table. Moreover, the instruments can be configured from the dropdown menu. The table also shows the most important information for each probe. The colour at the start of each row indicates the probe's operation mode. The history of measurements can be accessed by selecting a probe from the table. This also shows possible MVS nodes that are linked to the probe, that is, the measurements that are automatically updated to the MVS. In addition, the dropdown menu enables the user to configure the probe remotely, remove the probe from the instrument, meaning unload the probe, and link the probe to an MVS node. The configuration modal for a probe is shown in Figure 16.

Figure 16. Modal for configuring probes.

4.2. Functioning of the HSIS

The instruments use a separate configuration file to store configurations and parameters for each SUI. The configuration file overwrites instrument's default settings. Therefore, the instruments and the probes can be reused in other systems (R15) by excluding the configuration file. Figure 17 shows the file structure. The settings section contains the instrument-related configuration. In this section, the Universally Unique Identifier (UUID) [46], name, and instrument key are for the instrument. The probe-specific sections are named after the Secure Hash Algorithm 2 (SHA-256) hash function of the probe file. Each of the probe-specific sections encompasses the UUID, its name, and a description of the probe and the type of measurement. Moreover, the sections include run-time parameters such as the measurement frequency, possible dependency rule, file name, and operation mode. The configurations are loaded in two parts. Firstly, the *controller* loads the setting section. Secondly, the *probe loader* loads the probe-specific configurations.

```

[settings]
url = http://127.0.0.1:6300/
key = 211Gt8EONwa0MATFwejA6wB0baVOqkd1
name = A test instrument
uuid = b24fe7b5-00ed-4c2c-ace9-62f7bfef2767

[288273107982fa649fa419c5298527dc83207e840ff7df647e7431645c59823c]
mode = 2
name = Monitor login times
filename = login_time.py
description = Return timestamp for each login
dependency =
uuid = e6129834-0b6a-4bbb-a5d2-f8fe68e67fff
interval = 0
type = STR

[77f496bdddca84b779cbdead05fd06ae82614f2e297825891b05e71fc7c67e31]
name = Antivirus software name

```

Figure 17. Structure of configuration file.

The controller is initialised first to load all the configurations under the *settings* section. If the configuration file is missing, the instrument starts with the default settings and creates a new configuration file. The other components are started once the controller has finished loading the configurations.

Another component that uses the configuration file is the *probe loader*. The *probe loader* stores probe-specific information in the configuration file. This enables the *probe loader* to load previous settings, but it also eliminates the need to modify the probe file. The probe file stores only the initial configurations that are overwritten by the ones in the configuration file. Thus, the initial configurations are loaded only if the configuration file does not have a section for the probe. An example of a probe is shown in Figure 18.

```

import subprocess

name = "Antivirus software name"
description = "The name of the AV software installed"
type = "STR"
interval = 10

# "Normal mode" == 2, "Run once" == 1, "Manual" == 0
mode = 2

def run():
    pout = subprocess.Popen(
        'WMIC /Node:localhost /Namespace:\\\\root\SecurityCenter2 Path '
        'AntiVirusProduct Get displayName /Format:List',
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        shell=True).communicate()[0].strip()

    return pout.decode('utf-8').split("=")[1]

```

Figure 18. Example of a probe used by the instruments.

Three different criteria are evaluated before a probe is operated: the *operation mode* of the probe, the *dependency rule*, and the *measurement frequency*. The evaluation process is illustrated in Figure 19. The *mode* is used to control how many times the probe is operated, and the *measurement frequency* determines how often the probe is operated, such as once every second or once in a day. Moreover, the *dependency rule* allows the instrument adapt to changes in the environment by limiting the operation of probes to certain situations.

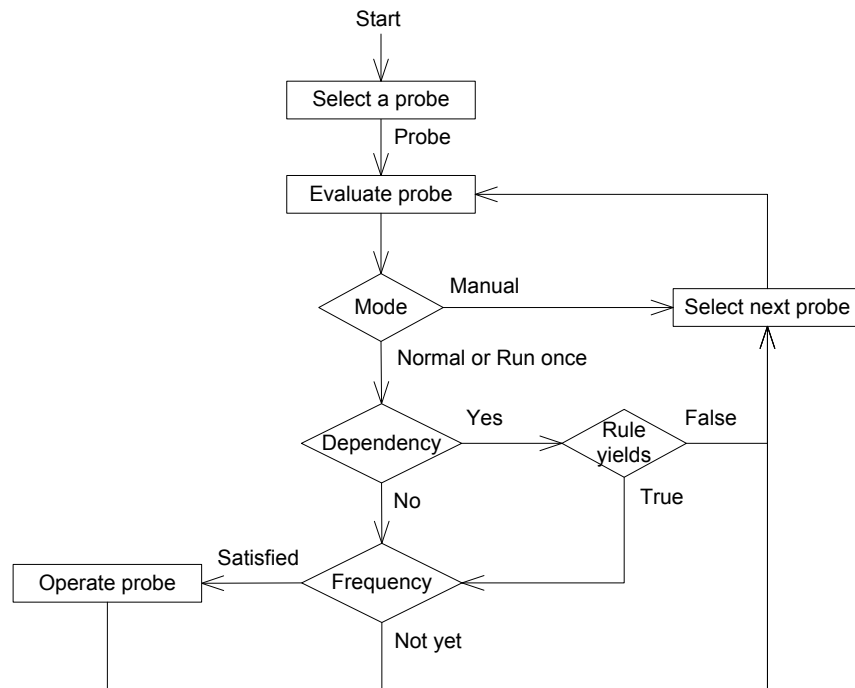


Figure 19. The steps in evaluating a probe.

The first criterion is the mode of the probe. There are three different modes for probes: *normal*, *run once*, and *manual*. Probes in *normal* mode are operated normally. In addition, the operation of a probe can be limited to only once. Lastly, probes in *manual* mode are not operated. This allows the instruments to preload probes in the instrument without actually operating them.

The adaptiveness aspect of the instrument comes from the *dependency rule* (R05). For instance, resource-intensive probes are operated only when it is necessary. The following notation is used for the dependency rule:

IF {PROBE_UUID | PROBE_FILENAME}
 IS (NOT) {EQUAL TO | LARGER THAN | SMALLER THAN} [VALUE]
 (AND IF {PROBE ...})

Compared to a traditional probe, which is an independent probe, the operation of the *adaptive* probe depends on the measurement value of the *main* probe, meaning whether the dependency rule yields true or false. *Adaptive* probes adapt to the environment based on the data gathered by the *main* probe. The first entry defines the *main* probe, on which the *adaptive* probe depends. *Adaptive* probes can depend on the UUID or on the file name of the *main* probe. By using the UUID as the reference,

adaptive probes are not operated if the *main* probe file is modified. This forces users to verify the content of the main probe. On the other hand, reusability of the probes can be enhanced by using the file name as a reference. However, the instrument will not be able to distinguish probes with the same file name. The second field defines how the main probe's measurement values are compared to the threshold value. There are three different ways to compare the values: *equal to*, *larger than*, and *smaller than*. The *equal to* option checks whether the values are identical or not. On the other hand, the other two options require that the *main* probe's most recent measurement value is larger or smaller than the set threshold value. Moreover, the comparison can be inverted by adding the 'NOT' parameter. Multiple dependency rules can be combined by adding the keyword 'AND' between each dependency rule. *Adaptive* probes are operated once the dependency rule yields true.

The dependency rule can be used to gather additional data based on the situation. For instance, adaptive probes can be set to gather network packets from a certain port once the main probe has detected malicious actions from that specific port.

The probe is put in the task queue once all the criteria have been satisfied. The available workers will pick their next task from the queue and operate the probes to obtain the measurement values. Probes that are put in the queue first are operated first, in the so-called first in, first out (FIFO) principle. Once finished, the results are added to the result queue, from which the *probe operator* collects the results. Each worker runs in a separate process to ensure that probes that take a longer time to operate do not prevent other probes from operating. The sequence diagram shown in Figure 20 demonstrates the process with two probes.

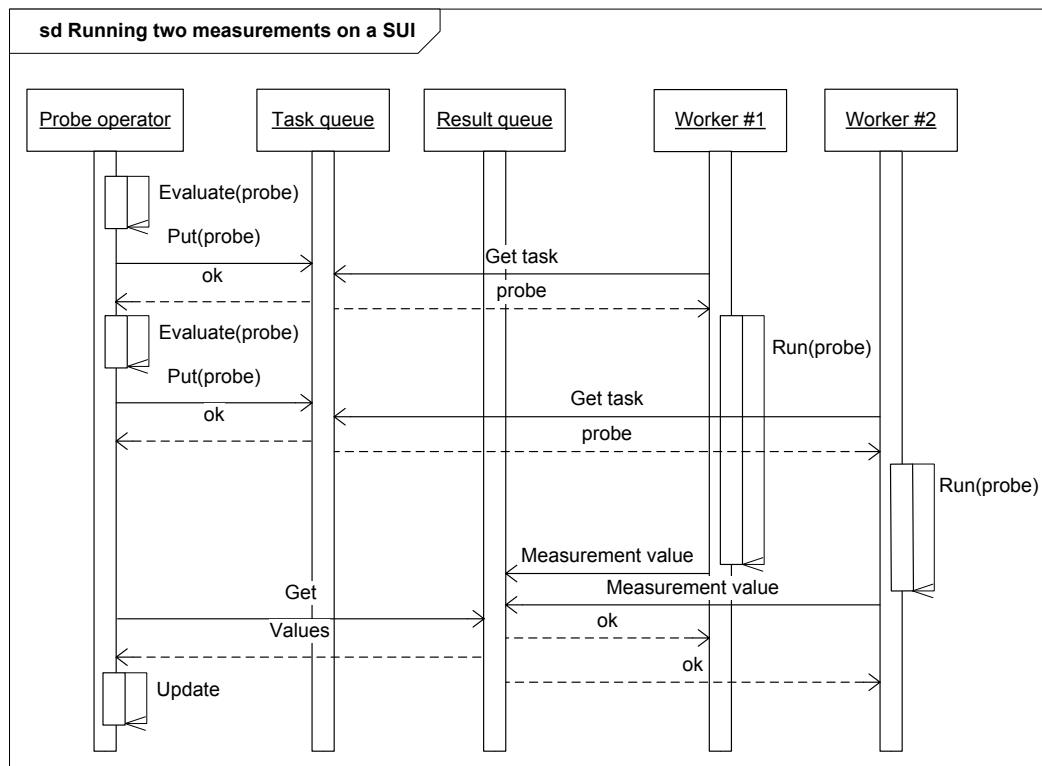


Figure 20. The operation and relation between the workers, the queues, and the probe operator.

The *forwarder* component enables the instrument to communicate with Manager. The process starts by constructing the output *message* that is sent to Manager in an interval. The *message* is structured the same way for all instruments (R03) and it contains information on the measurement values and the common attributes for the instrument and the probes (R07). The different elements of the *message* are shown in Figure 21. The message is divided into two sections: instrument- and probe-related information. Firstly, the attributes of the instrument are described. In addition, a timestamp is included to indicate when the message was prepared. Secondly, the attributes of the probe are described. As described above, an instrument may have one or more probes, and thus, each probe has a separate field in the message. The probe-related information is described in Table 5.

In addition, the run-time parameters are also included, namely the measurement frequency, mode, and dependency rule, for Dashboard users to know the current parameters. The measurement field contains multiple measurement values to ensure that measurements that have been conducted between the update intervals are correctly included in the message. Timestamps are formatted according to the ISO 8610 standard [47], which includes the date, time, and time zone.

```
{
  "uuid": "0c1980c2-95d8-4ac1-b74c-b20a096a26d9",
  "name": "This is a test instrument",
  "os": "WINDOWS",
  "timestamp": "2016-01-05T11:10:53+02:00",
  "probes": [{
    "uuid": "a3e25861-f2e5-4470-a04b-10e45c7b6959",
    "name": "Antivirus software name",
    "description": "The name of the AV software installed",
    "interval": 25,
    "mode": 2,
    "type": "STR",
    "measurement": [{
      "value": "F-Secure Client Security 11.60",
      "timestamp": "2016-01-05T11:59:40+02:00"
    }]
  }]
}
```

Figure 21. An example message with instrument information and measurement values of a probe.

Security-measuring instruments often gather sensitive and critical information from the SUI that can be beneficial to the attacker, and therefore, the message that is sent to Manager has to be protected (R12). AE provides confidentiality, integrity, and authenticity of messages.

The AE operation consists of two steps: i) encrypting the plaintext message, and ii) generating the message authentication code (MAC). The Galois/Counter Mode (GCM) is an AE process that carries out the two steps simultaneously. The HSIS uses GCM to protect messages, as it is not vulnerable to padding oracle attacks, as cipher block chaining (CBC) is. Moreover, the National Institute of Standards and Technology (NIST) recommends the GCM [57].

There are multiple steps in securing communication between the two subsystems. The GCM uses the Advanced Encryption Standard (AES) block cipher with a 128-bit symmetric key to encrypt the blocks used in AE. Therefore, both sides, namely Manager and the instrument, have to agree on a symmetric key that is used both to encrypt and to decrypt messages. The process starts by generating a unique *instrument key* in Dashboard. The instrument key is 192 bits long and it contains two separate keys: a 64-bit ID for the AES symmetric key and the 128-bit key itself. Manager and the instrument store the instrument key in the database and in the configuration file, respectively. The *instrument key* is unique to each instrument, and therefore, an ID is needed for Manager to find the correct symmetric key. Figure 22 shows a unique instrument key that is generated from Dashboard.

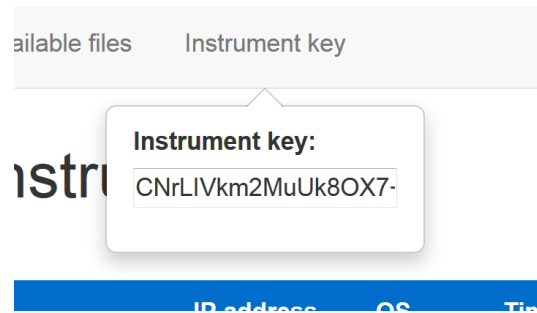


Figure 22. Generating a new instrument key from the Dashboard.

To encrypt messages, the *forwarder* generates a unique initialisation vector (IV), or nonce, for the GCM. Identical messages will yield in the same ciphertext without the IV. In GCM, the blocks are numbered sequentially starting from the IV. The numbered blocks are then encrypted using the AES block cipher, followed by XORing the encrypted blocks with the plaintext message blocks to produce the final ciphertext. The last step in the GCM is to calculate the MAC from the final ciphertext (the encrypt-then-MAC approach). Once the final ciphertext has been constructed, the MAC (tag), IV, and identifier are appended to the *envelope*. Thus, the *envelope* contains all the necessary information to decrypt the ciphertext, given that the recipient has the correct symmetric key that is indicated by the ID. Figure 23 visualises the AE process and the structure of the *envelope*.

Once finished, the *forwarder* sends the *envelope* to Manager. The decryption process starts by first separating the ID from the envelope. The ID is used to find the correct symmetric key. The ciphertext is decrypted using the correct key and the IV. During the decryption process, the authenticity and integrity of the message are verified. Lastly, the plaintext message is passed to the *database* interface, where it is further processed, and the information contained in the message is permanently stored in the database.

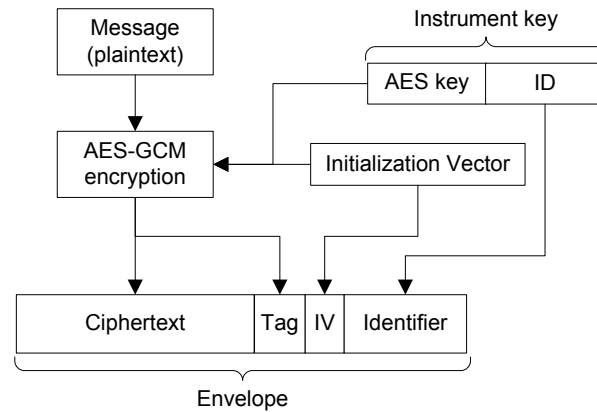


Figure 23. The structure of an envelope.

4.3. Implementation testing

The possible use cases of the HSIS are reflected in the test cases. The test cases are security auditing, adaptive measuring, and continuous monitoring of attributes. The HSIS capabilities to perform in these cases are demonstrated with the test cases. Moreover, the test cases are used to verify that requirements for the HSIS are satisfied.

In the first case, a system is audited and the results are shown in MVS. In the second test case, the measurements are not conducted until an external tool has identified possible security issues. In the last test case, the SUI is continuously monitored for security issues.

4.3.1. Test case: security audit visualisation with the MVS

Security auditing is an evaluation process to determine the security of an organisation's information systems by determining how well it satisfies a set of criteria. Moreover, auditing can be used to see if there has been any progress compared to the previous audit. In this test case, the auditor performs the security audit on two systems running the Windows operating system.

The test case demonstrates the capability of the HSIS to perform security auditing. The HSIS capability to forward the measurement values to third-party software is demonstrated by sending the results to the MVS. The MVS is visualisation software for modelling security concepts and security measurements. Finally, the reusability and the portability of the HSIS are validated by performing the same measurements on the two systems.

The measurements for the case are chosen from the CIS security audit benchmark for Windows 7. Nine different measurements are conducted on the SUI for demonstration purposes. The probes used in this audit test case are shown in Table 8 with a short description.

The HSIS and the MVS are stored in a clean USB drive. The drive contains all the required tools to perform an audit on the systems. Moreover, the instrument is

preloaded with the necessary probes to conduct the measurements. The measurement values are normalised between zero and one, due to the design of the MVS.

Table 8. List of probes that are used to audit the Windows system.

Probe name	Description	Values
Windows update status	Automatic updates ensure that the system is up to date.	1: On 0: Off
Firewall status	Firewall protects the system from network attacks.	1: On 0: Off
Screensaver status	Ensures that the system is locked if the user has forgotten log out.	1: On 0: Off
Require password for screensaver	Same as above.	1: True 0: False
Antivirus status	Active anti-virus protects the system from malicious software.	1: On 0: Off
Number of spyware instances left in the system	Spyware steals confidential information from the system.	1: Zero spyware instances 0: One or more spyware instances
Number of viruses left in the system	Viruses spread malicious software from computer to computer. Viruses can spread spyware.	1: Zero viruses 0: One or more viruses
Password age	Old passwords are more vulnerable to attack.	1: Equal to or less than 30 days old 0.5: Between 30 and 60 days old 0: Equal to or more than 60 days old
Minimum password length	Longer passwords are harder to guess.	1: Equal to or more than 13 characters long 0.5: Between 13 and 9 characters long 0: Equal to or less than 9 characters long

The audit is started by plugging the USB drive into the audited system. Firstly, the MVS and Manager are started. Secondly, the instrument is launched to operate the probes that conduct the measurements. The audit results are automatically updated to Manager from the instrument. Lastly, the results are forwarded to the MVS by

linking the probes to the correct MVS nodes from Dashboard. Figure 24 shows the MVS visualisation of the audit results for the first system.

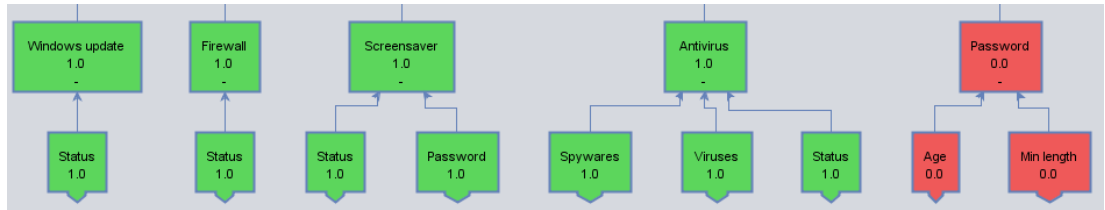


Figure 24. Audit results for the first Windows system.

The second system is similar to the first one, in that both run Windows 7, and thus, the same setup can be used (R15). Once the USB drive is plugged into the SUI, and the MVS and the HSIS are started, the data is automatically forwarded to the MVS. Figure 25 visualises the audit results for the second system.

The HSIS is portable, and it can be reused to security audit multiple similar systems. Moreover, it enables the auditor to perform customised measurements on each SUI. In a real-world scenario, more measurements are needed before any conclusion can be drawn.

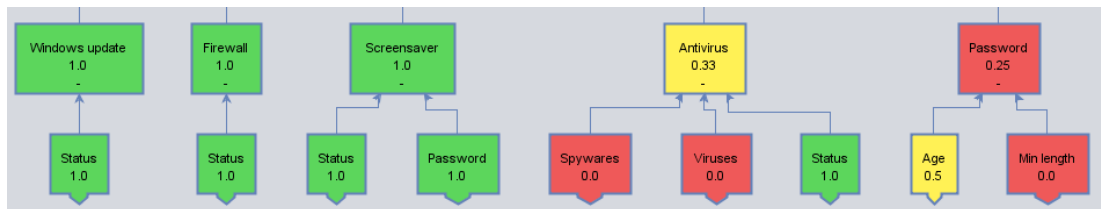


Figure 25. Audit results for the second Windows system.

4.3.2. Test case: adaptive auditing with an external tool

In the second case, a system running Ubuntu 14.04 LTS is security audited using the third-party tool *Lynis*. The *Lynis* tool scans the system for configuration errors and security issues. Moreover, it is capable of determining system-specific configurations, such as installed packages and operating system type. The *Lynis* tool acts as a probe, meaning that it gathers information from the SUI, and the HSIS is used to transmit the results from the SUI to Manager. The adaptiveness of the probe is validated by first operating the *Lynis* and then the results are analysed by the *adaptive* probe once the *Lynis* has finished its scan.

In this test case, the instrument operates normally in the Ubuntu system. Available probes are listed at the bottom of Dashboard, and the probes are remotely loaded into the instrument by selecting the correct probes from the list and assigning them to the instrument. Figure 26 shows the modal for assigning a probe to an instrument.

Firstly, the *main* probe that operates the *Lynis* tool is remotely loaded, followed by loading the *adaptive* probes into the instrument. The *adaptive* probes have a dependency rule that depends on the *main* probe, and thus, they will not be operated

unless the *main* probe has finished, that is, when the *Lynis* scan results are available. This rule prevents the *adaptive* probes from analysing old or unfinished scan results.

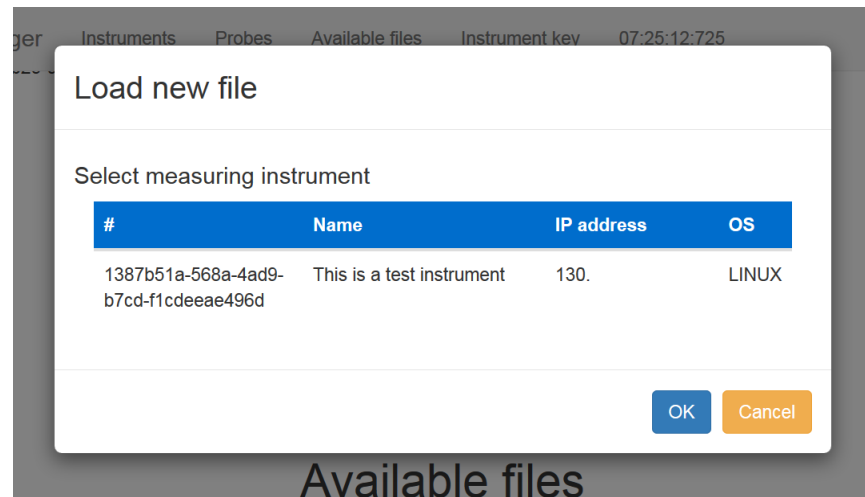


Figure 26. Modal for assigning a probe to an instrument.

The *main* probe automatically starts *Lynis* on the Ubuntu system. Once *Lynis* has finished, the scan results are stored in a log file. Figure 27 visualises the *Lynis* scan results. The *adaptive* probes are operated once the *main* probe has returned the value ‘True’, indicating that the scan results can be safely analysed.

The first *adaptive* probe checks from the scan results whether antivirus is installed in the system or not. The second one inspects the results for the status *auditd* [48]. Figure 28 shows the results for the test case. The first probe in the table is the *main* probe. The first *adaptive* probe returned zero, indicating that *auditd* is turned off and action should be taken to fix it. The second *adaptive* probe returned one, signifying that antivirus is installed in the audited system.

```

- Checking atd status [ NOT RUNNING ]

[+] Accounting
-----
- Checking accounting information... [ NOT FOUND ]
- Checking auditd [ NOT FOUND ]

[+] Time and Synchronization
-----
- Checking running NTP daemon (ntpd)... [ NOT FOUND ]
- Checking running NTP daemon (timed)... [ NOT FOUND ]
- Checking running NTP daemon (dnptd)... [ NOT FOUND ]
- Checking NTP client in crontab file (/etc/anacrontab)... [ NOT FOUND ]
- Checking NTP client in crontab file (/etc/crontab)... [ NOT FOUND ]
- Checking NTP client in cron.d files... [ NOT FOUND ]
- Checking event based ntpdate (if-up)... [ FOUND ]
- Checking for a running NTP daemon or client... [ OK ]

[+] Cryptography
-----
- Checking SSL certificate expiration... [ WARNING ]

[+] Virtualization
-----

```

Figure 27. *Lynis* scan results.

Probes

#	Name	Description	Type	Value	Timestamp	
4d5ed4fd-2db7-45e8-82c3-7f32aaca32cf	Lynis operator	Starts the lynis with parameters -c --cronjob	BLN	True	2016-02-22 11:40:09.504	▼
08ef02f8-95a8-4bd1-be7a-e4eba368ac74	Antivirus status HRDN-7230	Analyzes the scan results for HRDN-7230. 1: AV installed, 0: AV was not found	INT	0	2016-02-22 11:39:44.395	▼
09263d1a-6723-473b-9b25-96be7f5a5bd5	auditd status ACCT-9628	Analyzes the scan results for ACCT-9628. 1: auditd is on, 2: auditd is off	INT	1	2016-02-22 11:39:55.414	▼

Figure 28. *Lynis* scan results analysed by the HSIS.

4.3.3. Test case: real-time security monitoring

In the last test case, the HSIS is used to monitor attributes from a system running Windows 7. Continuous monitoring enables the HSIS to detect modifications made to the system in real time.

Multiple critical aspects of the SUI can be monitored in a real-world scenario. For demonstration purposes, three critical attributes are monitored in real time.

The probes are used to monitor user login times, the integrity of a critical folder, and the status of F-Secure SAFE [50] antivirus. Similarly to the second test case, the system already has an instrument running, and therefore, probes are remotely loaded into the SUI.

The first probe actively monitors for login times. The probe returns a timestamp every time someone logs into the system. The second probe observes a critical folder for changes. The probe monitors the folder for newly added or removed files. Lastly, the operation of F-Secure SAFE is monitored.

To validate the operation of the probes, a fictive intruder accesses the system. Figure 29 visualises the results of accessing the system. The first probe logged a login attempt at 14:37:49. Twelve seconds later, the second probe detected that F-Secure antivirus has been turned off. In addition, the last probe detected a new file 'MALWARE.BAT' in the critical folder. In this scenario, it can be concluded from the observed information that the SUI is in a critical state and immediate action should be taken to prevent the intruder from causing further damage to the system.

Probes

#	Name	Description	Type	Value	Timestamp	
f63721c3-102b-41a5-b454-b6e4bc1e87dd	Monitor login times	Returns the timestamp	STR	2016-03-08T07:36:35+02:00	2016-03-08 07:36:35.265	▼
fd5d05de-50c4-499c-aaf0-adb93a8cbe74	Monitors for folder integrity	Monitor for changes in a folder	STR	NEW: 'malware.bat'	2016-03-08 07:37:01.984	▼
0f9875b1-602e-4ba2-8dd1-a8a4c2d7728f	F-Secure AV status	Checks if the AV is turned on	BLN	FALSE	2016-03-08 07:36:47.936	▼

Figure 29. The results of the third test case.

4.3.4. Fulfilling the requirements

In Chapter 3.2, requirements were set for the HSIS. Firstly, Table 9 describes how the functional requirements are satisfied. Secondly, Table 10 presents the results for the NFR. An 'OK' status indicates that the requirements have been fully satisfied. A 'PARTLY' status indicates that some aspects of requirements have been fulfilled. The comment column provides additional information.

Table 9. Test results for the functional requirements.

R. ID	Description	Status	Comments
R01	Data gathering	OK	
R02	Measurement frequency	OK	
R03	Uniform message format	OK	The output messages have the same fields and structure.
R04	Forwarding the results	OK	The requirement is demonstrated with the MVS.
R05	Run-time adaptation	OK	The dependency rules allow the instrument to adapt the operation of the probes.
R06	Manageable and configurable remotely	OK	Instructions enable remote management and configuration. Moreover, Dashboard is accessible from mobile devices.

Table 10. Test results for the NFR requirements.

R. ID	Description	Status	Comments
R07	Common attributes	OK	Common attributes are stored in the database and visualised in Dashboard. The attributes distinguish individual instruments and probes.
R08	Modularity	OK	The instrument uses multiple independent components to operate.
R09	Non-intrusiveness	PARTLY	The resource usage was not significant during the test cases. However, further observations are needed to verify the claim.
R10	System isolation	PARTLY	Failure in the HSIS did not affect the reliability of the SUI. However, the failure was not forcefully induced.
R11	Highly dependable	PARTLY	The operation of the HSIS in extreme situations was not tested.
R12	Security	PARTLY	The measurements are protected with the AE. However, local attacks on the HSIS are still plausible.
R13	Availability	OK	The measurement values (and other information) are available from Manager regardless of the instrument's online status.
R14	OS independent	OK	The instrument operated in both Windows and Linux operating systems.
R15	Reusable	OK	The instruments and probes are not tied to the SUI. Run-time settings and parameters are stored in a separate configuration file.
R16	Custom measurements	OK	Users can create probes to conduct specific measurements. The instrument dynamically loads the probes, and thus, the probes are not hard-coded in the instrument.

5. DISCUSSION

It is crucial for building SA that security data is gathered from the SUIs. The HSIS helps administrators in performing measurements in multiple host systems. Moreover, the results from the security measurements can be forwarded to third-party software, such as the MVS. To facilitate security measurements, the HSIS provides common functions of security-measuring instruments and offers an interface to manage and control the instruments and the probes. The provided interfaces are demonstrated using Dashboard. For instance, users can remotely change settings for instruments or view the measurement values with a mobile device, such as with a mobile phone or a tablet.

For the instruments to operate in a dynamic environment, they adapt to environmental changes by using dependency rules for the probes. This enables the instrument to have greater control over when to operate the probes. For example, the dependency rule is used in the second test case to analyse the Lynis scan results once they are available. The dependency rules prepare the instrument for different changes in the environment.

Additional data provide better understanding of situations. For instance, intrusion detection systems tend to have high false-positive and false-negative rates, especially in the case of zero day attacks [51]. The HSIS helps users in deploying new measurements on the SUI to gather additional information on the situation.

The usability of the HSIS is demonstrated in three different test scenarios. The instruments gathered data from the SUI and securely updated the results to Manager.

Finally, the utilisation of the HSIS is not limited to security measurement. It can be used to measure other attributes that do not contribute to security. However, the thesis focuses on the security aspects of the SUI, and therefore, the test cases were built accordingly.

In the first subchapter, the presented implementation is compared to the presented similar solutions. In the second subchapter, discussion on the limitations of the HSIS is presented, and lastly, future work is briefly discussed.

5.1. Comparison to other similar solutions

Unlike other solutions, the HSIS adapts to the dynamic environment. This ensures that measurement needs are satisfied in all situations. Moreover, compared to the Nessus plugins, the HSIS support probes are written in Python. Python is better documented than the NASL, and it is well supported by the community, so that there are multiple modules and libraries that can be utilised in creating custom probes. Therefore, Python offers a great platform on which to build probes. However, the instruments are not bundled together with a large number of probes that gather the same amount of data as the existing solutions, but the HSIS supports user-created probes to conduct specific measurements. Moreover, the instruments can be remotely configured to suit the needs of a specific host system. Finally, the HSIS can be future developed to meet the needs of VTT's cybersecurity team.

The HSIS uses a custom message format to overcome some of the disadvantages of the most common format, the syslog format (RFC-3164) [55]. Major problems with the specific syslog include incomplete time format, namely the lack of a time zone and year stamp, and the textual structure of the message. The newer syslog

format (RFC-5424) [56] overcomes some of the shortcomings. However, the textual structure remains, and thus, information that has a rich, structured format cannot be preserved when using the syslog format.

Compared to existing solutions, the HSIS is portable and versatile in terms of conducting security measurements. The HSIS does not require any modification or installation in the system. Moreover, the instruments are reusable, and the whole HSIS can be launched from a USB drive.

5.2. Limitations

One major decision is made regarding how the instrument and Manager communicate. In the HSIS, the instrument can only initialise the communication, meaning that Manager cannot directly communicate with the instruments. This kind of setup eliminates the need for configuration on the host side, such as by opening unnecessary ports, as this may introduce new vulnerabilities. However, it presents additional delays in the communication between the two subsystems, as Manager does not have control when the instrument establishes the connection. The delay can be reduced by increasing the frequency with which the instrument connects to Manager. However, this will increase the load on the network.

In addition, loading new probes into instruments proved to be time-consuming due to the way the instruments and Manager communicate. It took several seconds before the results were updated to Manager. Such a delay can be intolerable in time-critical situations.

The instrument uses multiple processes to simultaneously operate the probes. However, the instrument struggles when there are many attributes that are measured in real time. Unlike conducting a single measurement, real-time measurement uses the whole process. This reduces the number of available processes, or workers, for other probes to operate, and thus, may cause a situation where the instrument is unable to operate all the probes with the given measurement frequency.

Finally, the secure aspect of the system is limited to the communication between the instruments and the managing unit, Manager. Moreover, accessing Dashboard does not currently require any authentication. Some aspects of the HSIS are secured; however, the main purposes of the HSIS are to demonstrate adaptive security measurement and to present the common functions of security measurements. Therefore, additional protection is needed to deter attacks on the HSIS.

5.3. Future work

The HSIS is currently under development. The focus of the HSIS has been on implementing basic features to conduct adaptive security measurement. The next phase in the development is overcoming the shortcomings and limitations described above.

Currently, the probes use a very basic adaptation. The current adaptation is determined by the dependency rules: the probes can depend only on the probes that are operated by the same instrument, and multiple rules can only be combined using the 'AND' operator. However, in the future, the rules can be combined using other operators as well, such as 'OR', 'XOR', and 'NAND', to add new possible combinations in the rules. With more specific dependency rules, the instruments will

be more capable of adapting to situations. In addition, in the future, *adaptive* probes can depend on *main* probes that are located in other host systems. Manager-wide dependency rules enable the instruments to react to situations that are currently affecting other systems.

Machine learning will be examined in the future, along with its implementation in the HSIS. For instance, probes could be turned on and off dynamically, without user interference. The instruments would also be capable of automatically adjusting themselves and the probes to different scenarios by changing run-time parameters, such as the measurement frequency of the probes to optimise resource usage in the SUI.

Currently, the instruments can download new probes and other files from Manager. There are no restrictions on what files can be downloaded by the instrument, as the instrument is capable of distinguishing probe files from other file types. However, the process is limited to single files, so that folders cannot be downloaded. This will be addressed in the future, as it would then enable instruments to download complete software suites, which act as a probe that can be used by the HSIS probes.

In the design, the possibility of integrating the HSIS with the MVS is taken into account by implementing Manager with the same programming language, namely Java. Dashboard is used to demonstrate the interface that is later accessible from the MVS user interface, meaning that both systems can be managed from one single user interface. This will greatly increase usability and automate some of the processes that are currently cumbersome, such as linking probes to the MVS nodes.

Additional testing is needed to verify some of the NFR. Resource usage will be monitored in more detail, and the isolation of the HSIS from the SUI will be tested using fuzz testing. Moreover, the HSIS will be tested in a more unstable environment, to see the effect on the HSIS. Lastly, the security aspects of the HSIS will be extended to combat attacks that target Manager and the instruments themselves.

Finally, the possible utilisation of the HSIS in other fields and scenarios will be examined. This will greatly extend the usability of the HSIS.

6. CONCLUSION

Gaining SA is essential in making proper security countermeasures. However, gathering the information requires multiple measurements from the systems in the network. Moreover, managing large numbers of instruments can cause problems. Existing solutions offer little support for custom measurements. Moreover, the solutions lack adaptive security instrumentation.

The HSIS is the reference implementation of the security-measuring framework that overcomes these issues. The main purpose of the HSIS is to support users in conducting customised adaptive security measurements on a host system. This is achieved by using two subsystems: the *instruments* and *Manager*. The instruments are host-based information-gathering units that update the results to Manager. On the other hand, Manager commands the instruments and stores the information sent by the instruments in its database. The user can view and manage instruments by accessing the web application Dashboard.

Dashboard visualises the information sent by the instruments in one single page. It displays the most recent information on the instruments and the measurements in two separate tables. The user can remotely reconfigure the instruments and the probes from Dashboard. Moreover, Dashboard enables users to remotely view and load probes into instruments.

Due to the design, the HSIS is very versatile in terms of measurements. The instruments dynamically load and operate the *probes* that are responsible for the actual data-gathering. The instrument itself is capable of processing different sorts of data gathered by the probes. Moreover, the message format used by the HSIS preserves the structure of the measurement values.

Another purpose of the HSIS is to offer the collected results to analysing entities. Manager is capable of updating the measurement values for external analysing entities for visualisation and analysis. In the test cases, measurements are forwarded to the MVS for visualisation. This, in turn, helps the user to build SA, and thus aid users in decision-making.

The focus of this thesis is to promote the process of conducting adaptive and custom security measurements. The common and non-common functions are identified from the security-measuring process. The common ones are implemented by the HSIS. Finally, the requirements set for the thesis are validated in three different scenarios.

The HSIS is currently under development, and future releases are expected. The test cases revealed shortcomings and cumbersome features that will be addressed in future versions. However, the current version demonstrates the basic operation of managing and conducting multiple measurements on a host system. Moreover, the HSIS enables its users to design probes for specific needs, and therefore, the probes can gather even the most specific data. Finally, the HSIS demonstrates the basic principle of adaptive security measurement.

Integrating the HSIS with the MVS in the future could be an invaluable combination for security experts to design, implement, and visualise security information.

7. REFERENCES

- [1] Beaudoin L., Froh M., Gregoire M. & Lefebvre J. (2006) Computer network defence situational awareness information. Proc. 2006 IEEE Military Communications Conference, Washington, DC, 1-7.
- [2] Thomsen S. (Accessed 10.9.2015) Ashley Madison was a bunch of dudes talking to each other, data analysis suggests, Business Insider UK. URL: <http://uk.businessinsider.com/cheating-affair-website-ashley-madison-hacked-user-data-leaked-2015-7?r=US&IR=T>.
- [3] Chung E. (Accessed 10.9.2015) PlayStation data breach deemed in 'top 5 ever', CBC News. URL: <http://www.cbc.ca/news/technology/playstation-data-breach-deemed-in-top-5-ever-1.1059548>.
- [4] Wang C. & Wulf A. (1997) Towards a framework for security measurement. Proc. 1997 20th National Information Systems Security Conference. Baltimore, Maryland, 522-533.
- [5] Brotky W. & Hinson G. (2012) Pragmatic security metrics. Boca Raton, Florida, CRC Press.
- [6] Bayuk J. & Mostashari A. (2011) Measuring systems security. Systems Engineering 16(1): 1-14. DOI: 10.1002/sys.21211.
- [7] ISO 27000:2014 (2014) Information technology – Security technique – Information security management - Measurement, International Organization for Standardization.
- [8] ISO/IEC 27032:2012 (2012) Information technology – Security techniques – Guidelines for cybersecurity, International Organization for Standardization.
- [9] Rossouw von Solms, Joha van Niekerk (2013) From information security to cyber security. Computer & Security 38: 97-102. DOI: 10.1016/j.cose.2013.04.004.
- [10] Kissel R. (2013) Glossary of Key Information Security Terms. National Institute of Standards and Technology, NIST.
- [11] Evesti A. (2013) Adaptive security in smart spaces. Doctoral thesis. University of Oulu, Department of Computer Science and Engineering.
- [12] Anderson R. (2008) Security engineering: a guide to build dependable distributed systems, second edition. Indiana, Wiley Publishing Inc.
- [13] Radack S. (2009) The system development life cycle (SDLC). National Institute of Standards and Technology, NIST.

- [14] Savola R. (2009) A Security metrics taxonomization model for software-intensive systems. *Journal of Information Processing Systems* 5(4): 197-206. DOI: 10.3745/JIPS.2009.5.4.197.
- [15] Radack S. (2010) Security metrics: measurements to support the continued development of information security technology. National Institute of Standards and Technology (NIST).
- [16] Pfleeger S. & Cunningham R. (2010) Why measuring security is hard. *IEEE Security & Privacy* 8(4): 46-54. DOI: 10.1109/MSP.2010.60.
- [17] Watt D. (Accessed 29.9.2015) So you can't pick the hits. Neither can anyone else. *The Washington Post*. URL: <http://www.washingtonpost.com/wp-dyn/content/article/2009/01/02/AR2009010202194.html>.
- [18] Savola R. (2010) On the feasibility of utilizing security metrics in software-intensive systems. *International Journal of Computer Science and Network Security* 10(1): 230-239.
- [19] Savola R. (2012) Strategies for security measurement objective decomposition. *Proc. 2012 Information Security for South Africa (IISA)*. Johannesburg, Gauteng, 1-8. DOI: 10.1109/ISSA.2012.6320434.
- [20] McHugh J. (2001) Quantitative measures of assurance: prophecy, process, or pipedream? *Proc. 2001 Workshop on Information Security System Scoring and Ranking (WISSSR)*, ASCA and MITRE. Williamsburg, Virginia.
- [21] McCallam D. (2001) The case against numerical measures of information assurance". *Proc. 2001 Workshop on Information Security System Scoring and Ranking (WISSSR)*, ASCA and MITRE. Williamsburg, Virginia.
- [22] Savola R. (2012) Quality of security metrics and measurements. *Computers & Security* 37: 78-90. DOI: 10.1016/j.cose.2013.05.002.
- [23] Jelen G. (2000) SSE-CMM Security metrics. NIST and CSSPAB Workshop. Washington, D.C..
- [24] García F., Bertoa M., Calero C., Vallecillo C., Ruíz F., Piattini M. & Genero M. (2005) Towards a consistent terminology for software measurement. *Information and Software technology* 48: 631-644. DOI: 10.1016/j.infsof.2005.07.001.
- [25] Evesti A., Savola R., Ovaska E. & Kuusijärvi J. (2011) The design, instantiation, and usage of information security measuring ontology. *The Second International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*: 1-9.

- [26] ISO 27004:2009 (2009) Information technology – Security technique – Information security management - Measurement, International Organization for Standardization.
- [27] Jaquith A. (2007) Security metrics: replacing fear, uncertainty, and doubt. Boston, Addison-Wesley.
- [28] Center for Internet security (CIS) benchmarks (Accessed 21.9.2015). URL: <https://benchmarks.cisecurity.org/>.
- [29] Common Vulnerabilities and Exposures (Accessed 2.9.2015), MITRE. URL: <https://cve.mitre.org/>.
- [30] Jajodia S., Liu P., Swarup V. & Wang C. (2010) Cyber SA: Situational Awareness for Cyber Defence, Cyber situational awareness: Issues and Research, Advanced in Information Security 46: 3-14. DOI: 10.1007/978-1-4419-0140-8.
- [31] Klein G., Tölle J. & Martini P. (2011) From detection to reaction - A holistic approach to cyber defense. Proc. 2011 Defense Science Research Conference and Expo (DSR). Singapore, 1-4. DOI: 10.1109/DSR.2011.6026824.
- [32] Wireshark (Accessed 9.2.2016) An open-source tool to analyse network packets, Wireshark Team URL: <https://www.wireshark.org/>.
- [33] Nmap (Accessed 9.2.2016) Network scanner that is used to discover hosts and services, Gordon Lyon. URL: <https://nmap.org/>.
- [34] Lynis (Accessed 9.2.2016) A security audit tool for Linux systems, Cisofy. URL: <https://cisofy.com/lynis/>.
- [35] Apache Derby (Accessed 7.3.2016) A relation database management system, Apache Software Foundation. URL: <https://db.apache.org/derby/>.
- [36] Kanstrén T., Savola R., Haddad S. & Hecker A. (2011) An adaptive and dependable distributed monitoring framework. International Journal on Advances in Security 4(1-2): 80-94.
- [37] Gamma E., Helm R., Johnson R. & Vlissides J. (1994) Design patterns: elements of reusable object-oriented software. Reading, Massachusetts, Addison-Wesley Publishing Company.
- [38] Requests (Accessed 9.2.2016) A HTTP library for Python. URL: <http://docs.python-requests.org/en/master/>.
- [39] Hypersonic 2 (H2) (Accessed 9.2.2016) A database engine. URL: <http://www.h2database.com>.

- [40] Microsoft SQL Server (Accessed 9.2.2016) Relational database management system, Microsoft. URL: <https://www.microsoft.com/fi-fi/server-cloud/products/sql-server/>.
- [41] MySQL (Accessed 9.2.2016) An open-source relational database management system, Oracle Corporation. URL: <https://www.mysql.com/>.
- [42] Oracle Database (Accessed 9.2.2016) An object-relation database management system, Oracle Corporation. URL: <https://www.oracle.com/database/index.html>.
- [43] Hypersonic 2 (H2) (Accessed 9.2.2016) Database engine performance comparisons. URL: <http://www.h2database.com/html/performance.html>.
- [44] Bootstrap (Accessed 9.2.2016) An open-source front-end framework. URL: <http://getbootstrap.com/>.
- [45] AngularJS (Accessed 9.2.2016) An open-source web application framework, Google. URL: <https://angularjs.org/>.
- [46] RFC-4122 (2005) A Universally unique Identifier (UUID) URN Namespace.
- [47] ISO 8601:2004 (2004) Data elements and interchange formats – Information interchange – Representation of dates and times, International Organization for Standardization.
- [48] Auditd (Accessed 9.2.2016) A tool to monitor to Linux system. URL: <http://people.redhat.com/sgrubb/audit/>.
- [49] Latvala O-M., Toivonen J., Kuusijärvi J. & Evesti A. (2014) A tool for security metrics modelling and visualization. Proc. ECSAW 2014 Proceedings of the 2014 European Conference on Software Architecture Workshops 3. DOI: 10.1145/2642803.2642806.
- [50] F-secure SAFE (Accessed 9.2.2016) Cross-platform antivirus software, F-Secure. URL: https://www.f-secure.com/fi_FI/web/home_fi/safe.
- [51] Stolfo S., Bellovin S. & Evans D. (2011) Measuring security. IEEE Security & Privacy 9(3): 60-65. DOI: 10.1109/MSP.2011.56.
- [52] Nessus (Accessed 9.2.2016) A vulnerability scanner, Tenable Network Security. URL: <http://www.tenable.com/products/nessus-vulnerability-scanner>.
- [53] Snare (Accessed 9.2.2016) A tool to collect audit log data, InterSect Alliance. URL: <https://www.intersectalliance.com/>.
- [54] Assuria Cybersense (Accessed 9.2.2016) A security configuration assurance and policy compliance solution, Assuria. URL: <http://www.assuria.com/cybersense-enterprise-scanner.html>.

- [55] RFC-3164 (2001) The BSD syslog Protocol.
- [56] RFC-5424 (2009) The Syslog protocol.
- [57] Dworkin M. (2007) Recommendation for block cipher mode of operation: Galois/Counter mode (GCM) and GMAC, National Institute of Standards and Technology (NIST).