Agentless Endpoint Security Monitoring Framework

by

Asem Ghaleb
B.Sc., Taiz University, 2007
M.Sc., King Fahd University of Petroleum and Minerals, 2016

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Agentless Endpoint Security Monitoring Framework

by

Asem Ghaleb
B.Sc., Taiz University, 2007
M.Sc., King Fahd University of Petroleum and Minerals, 2016

Supervisory Committee

Dr. Issa Traore, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai Sima, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Issa Traore, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai Sima, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Existing endpoint security monitors use agents that must be installed on every computing host or endpoint. However, as the number of monitored hosts increases, agents installation, configuration and maintenance become arduous and requires more efforts. Moreover, installed agents can increase the security threat footprint and several companies impose restrictions on using agents on every computing system. This work provides a generic agentless endpoint framework for security monitoring of computing systems. The computing hosts are accessed by the monitoring framework running on a central server. Since the monitoring framework is separate from the computing hosts for which the monitoring is being performed, the various security models of the framework can perform data retrieval and analysis without utilizing agents executing within the computing hosts. The monitoring framework retrieves transparently raw data from the monitored computing hosts that are then fed to the security modules integrated with the framework. These modules analyze the received data to perform security monitoring of the target computing hosts. As a use case, a real-time intrusion detection model has been implemented to detect abnormal behaviors on computing hosts based on the data collected using the introduced framework.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My mother Maryam and my family for their encouragement, prayers, love, and support through my journey.

My supervisor, Dr. Issa Traore for his unstinting support, mentoring, encouragement, guidance and for spending hours on my drafts.

My thesis committee for their worthy comments and suggestions.

ISOT lab colleagues, staff, and friends for their help and support.

As for the foam, it vanishes, [being] cast off; but as for that which benefits the people, it remains on the earth.
Quran surah Ar Ra'd 17 (QS 13: 17)

DEDICATION

To my mother Maryam, my father Abdo Esmail, and my brother Aimaan.

Chapter 1

Introduction

1.1 Context and Research Problem

The traditional way of monitoring computing hosts, referred to as agent-based approach, often requires the installation of agents software on the target hosts that periodically scan targeted hosts and collect data about the hosts or the applications running on hosts [9]. The collected data are either transferred to a management server for analysis or analyzed by a management software locally on the same host. However, this approach of monitoring suffers several scalability, security and performance drawbacks, and those drawbacks affect user acceptability. There is a need to install agent software and perform detailed configuration for each host [12], and the installed agents require continuing maintenance. In addition, the use of agents impacts deployment time on computing hosts, as agents should be installed and their updates should be thoroughly checked and applied before deployment on the hosts. Thus, hosts may get compromised if updates are not applied regularly and when new hosts are plugged in the network before having updated agents installed on them. With regards to security, the installed agents on hosts may involve vulnerabilities which increase the attack surface of the monitored hosts. Attackers can target the installed agents and their running services and take advantage of the privileges granted to the agents. Hence, compromising the agents enables attackers to get control of the hosts monitored by the agents. Another major drawback of agent-based approach appears when there are several virtual machines running on a computing host. In this case, there would be a need to install agents on every virtual instance besides the agent installed on the hosting machine itself. This could impact the functionality

and the performance of the computing hosts, as each agent running on every instance consumes some resources out of the total computing host resources.

An agentless approach, on the other hand, collects data from hosts without installing any agent on the computing hosts being monitored [9]. This makes agentless approach easy to manage than agent-based approach. However, to our knowledge limited research has been done in this area to this date, and almost all existing studies focused on collecting specific types of data agentlessly and designing models to detect specific attacks. There has been no study that addressed proposing generic agentless architecture that can be used to perform comprehensive collection of data needed to build numerous security models for detecting and protecting against various security attacks.

The objective of this thesis is to address the aforementioned challenges facing agent-based security monitoring and the limitations of existing agentless studies and propose a generic agentless framework for security monitoring of computing hosts. The monitoring process should be achieved in an efficient, low latency, scalable, and secure way by utilizing regular communication protocols that are often pre-installed and standard on most machines. The aim is to use the proposed framework for collecting host-based data and integrate it with anomaly-based intrusion detection models. The current thesis addresses the following research questions:

- **RQ1**: What is the scope of endpoints security monitoring that can be performed using the proposed generic agentless architecture?
- **RQ2**: How can agentless architecture address the scalablity, performance and security issues of the agent-based architectures?
- **RQ3**: What are the limitations of using agentless architecture for computing hosts monitoring?

1.2 Proposed Approach

To address the scalability, performance and security challenges of the existing agent-based security monitoring mechanisms discussed in the problem statement section, we propose a framework to monitor the computing hosts remotely without employing any agents running on the computing hosts. The monitoring framework is to be integrated with intrusion detection and anti-malware models and provide them with

data collected from the monitored hosts. These models are used for analyzing the received data to perform security monitoring of the target computing hosts.

The monitoring framework consists of an architecture that is scalable and can securely access any computing host via regular and standard protocols to perform all the functions that are necessary for host-based security monitoring without degrading the network performance. The SSH [5] protocol will be used in implementing a prototype of the proposed framework. The proposed framework is to be configured to run on a central server that has access to the computing systems supposed to be monitored by the framework.

The security monitoring can be performed by collecting data that can be used to scan for abnormal and malicious behavior of the file system, running processes, active network connections, Windows registry, and system resource usage, etc. Example of data collected by our framework include the following groupings:

- User activity data
- Process table
- File system
- Windows registry
- Network connections
- Resource consumption data

The variety of the data items that can be collected by the framework enables it to be used for scanning and detecting wide range of host-based attacks. For example, the logs of the user activity can be used to detect abnormal and malicious logins, password cracking, masquerading, and denial of service (DoS) attacks that target user accounts, to name a few. The data of the processes spawned can be used to detect couple of attacks. Large number of malware create processes when they are installed in the infected machines, and some other target injecting the running normal processes. The collected data of the processes spawned can help in detecting these kinds of malware, either during the infection or during the investigation process as part of the digital forensics process that investigates how an attack was launched or how a machine was infected. The data related to the file system activity form a useful source for detecting several malware and attacks. The abnormal behavior happening

in the file system when the machine is under infection can be used to detect the source of such behavior. A good example of this is the massive files change that happens when a ransomware starts its malicious work by encrypting enormous number of files on the targeted host.

As the majority of the released malware target hosts running Windows OS, and most of those malware do changes in Windows registry, the data collected from Windows Registry help in detecting number of malware and attacks. The established network connections and the network traffic are helpful in detecting large set of malware attacks, such as botnet, DoS attacks, etc. Monitoring the resources consumption helps not only for performance monitoring but for security monitoring as well, especially for detecting attacks that target system availability, such as DoS attacks.

The collected data from different sources could be combined together for performing security monitoring. For example, the detection of viruses based on the data related to the events of the file system, registry and running processes. Moreover, the framework can be used to collect other data, such as capturing data from the host memory that might be used to build a security defense model that detects malware based on the data captured from the host memory. To sum up, the data collected using the proposed framework and the type of the host-based attacks that can be covered using those data are not limited to the examples aforementioned. The framework is designed to be generic, and its functionality can be extended by end-users.

1.3 Thesis Contributions

The main contributions of this thesis are as follows.

- Proposing a generic and scalable agentless endpoint security monitoring framework.
- Study of the raw data that can be collected using the agentless architecture.
- Designing and implementing a prototype of agentless endpoint framework for security monitoring of computing systems.
- Introducing a real-time intrusion detection model that works on detecting abnormal behaviors based on the data collected using the proposed agentless architecture.

Part of the above contributions have been accepted for publication in a conference [17].

1.4 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2 provides a literature review of the related work in the areas of agentless security monitors and detectors. In addition to a review of related work in the filed of network forensics.

Chapter 3 introduces our agentless endpoint security monitoring framework along with its architecture, design and implementation details. This chapter provides an overview of the proposed techniques and components shaping the proposed framework.

Chapter 4 presents a security and efficiency evaluation of the proposed framework.

Chapter 5 presents a real-time intrusion detection use case.

Chapter 6 summarizes the contributions of the current work, and discusses possible research directions for future work.

Chapter 2

Literature Review

The concept of agentless monitoring has been used for building monitors and malware/intrusion detectors, especially in the cloud and virtual machines. In this chapter we review related research for agentless monitoring and present the findings in this field. In addition, we review the traditional monitoring frameworks and tools. Finally, we review related articles highlighting agent-based vs. agentless monitoring.

2.1 Agentless Monitors and Detectors

Berlin et al. [11] investigated the potential of using an agentless utility for detecting malicious endpoint behavior based on Windows audit logs as a supplement for the existing defense tools. They have setup a data collector to collect events of the file/registry's writes, deletes and executes, and processes spawned. The events are collected by running malicious and benign samples in a sandbox for a time-window of 4 minutes. A Logistic Regression (LR) model has been trained using the features extracted from the collected events. The model achieved 83% detection rate with 0.1% false positive rate. However, only offline test experiments have been used to evaluate the model. Unlike our framework, the proposed approach deals only with events related to file/registry's writes, deletes and executes, and processes spawned. The proposed models can not be expanded by users to integrate other audit logs.

Tang et al. [26] proposed an agentless antivirus system (VirtAV) for Antivirus protection on VMs based on in-memory signature scanning. To prevent attacks in the guest VM from reaching the antivirus system, in this work, the monitoring of events and detection of virus are offloaded to the hyeprvisor or virtual machine monitor

(VMM). VirtAV-engine searches for viruses' signatures by scanning the host memory for footprints of executable in guest VMs. A prototype has been implemented based on Qemu/KVM hypervisor. The evaluation experiments based on 3546 samples of viruses showed that the approach can find all the sample viruses, and acceptable overhead is introduced to the VM. However, the proposed system does not have generic functionality; it is designed only for detecting viruses based on in-memory signatures.

Brattstrom et al. [12] proposed scalable and agentless network monitoring system. The monitoring system combines a collector, a time series database and a dashboard running on 'plug-and-play'Pi. In their implementation, each Raspberry Pi is loaded with a Docker image of the system. The Pi uses Simple Network Management Protocol (SNMP) to poll the monitored network devices. The collected data is stored in the database and presented on Grafana dashboard. The authors claim that adding more Pi devices to the network supports the scalability of the system. The paper did not discuss the types of data that can be collected. The system just presents data analytics on the dashboard. Unlike our work, the data in the time series database is not to be accessed and used by other security defense models. The proposed system is not generic, as the extension of the polled data requires modification of the collector deployed on the Pi devices as part of the preconfigured Docker image.

An effort toward preventing vulnerabilities emerged due to the use of VM based antivirus software and reducing CPU and memory consumption is presented in [14]. In this work, Cui et al. proposed an agentless architecture for processes monitoring in OpenStack cloud platform. In the proposed architecture, the KVM kernel has been modified and security modules have been added on both computing and management nodes. Running processes on each VM are analyzed agentlessly using process-handling on the computing node at first, then new processes are sent to process-scanning module on the manager node through netlink. If a process is recorded as suspicious, it would be sent to the decision-making module on the manager node to decide whether to terminate the process or to keep it. However, the proposed architecture was designed only for monitoring processes on VMs in OpenStack cloud platform. Moreover, the architecture is not standalone and requires modification to the KVM kernel.

Ceilometer [13] is an OpenStack tool for cloud monitoring. It collects a number of metrics from the physical machines deployed on the cloud via agents. In addition, it retrieves metrics about the virtual machines from the control plane transparently (agentlessly) by interrogating the hypervisor. Ceilometer does not have real connec-

tivity with virtual machines due to the fact that it is not integrated with the data plane of the cloud, thereby limiting the number of metrics that can be retrieved from the VMs. Moreover, it does not support the collection of wide range of data that can be used for the purpose of security monitoring as we proposed in our work.

The limitations of Ceilometer, because of not being integrated with the data plane, were addressed by a monitoring solution for public cloud proposed by Gutierrez-Aguado et al. [18]. The proposed architecture supports transparent (agentless) monitoring of VMs and agent-based monitoring of physical machines, disks, and other resources. The VMs are monitored using transparent methods, such as metrics extracted directly from the hypervisor, ICMP connections, and UDP/TCP connections. The architecture is not dedicated for security monitoring and no focus is shown on what can be monitored using the proposed architecture.

Another related software is the security information and event management (SIEM) software [6]. SIEM works on aggregating and analyzing log data generated by systems, applications, network hardware and security devices such as firewalls, IDs and antivirus models. However, it is main functionality is based on aggregating log data from different log sources rather than collecting various raw data as performed by our proposed framework.

2.2 Traditional Monitoring Architectures for IT Infrastructure

Traditional monitoring architectures for IT infrastructure rely on the installation of agent software, responsible of collecting the data items, in the monitored endpoints. In this section, we review the distributed systems based forensic frameworks that focus on the networked environments, and we discuss how the proposed frameworks can be implemented agentlessly. In addition, we review the most common traditional IT infrastructure monitoring tools.

2.2.0.1 Network Forensic Frameworks

The generic process of computer forensics goes through several steps that involve preparation, identification, extraction, interpretation or analysis, documentation and finally presentation. Several frameworks have been proposed by the research community for computer forensics. Similar to our work, the proposed frameworks for

automating the network forensic process involve components that target the data collection and some other components that conduct the investigation and analysis. The data are collected from several networked devices. However, all the proposed frameworks are based on agents for collecting and analyzing data and evidence.

Shanmugasundaram et al. [25] proposed a distributed logging framework (ForNet) to facilitate the digital forensics over large networks. FortNet architecture involves two core components, SynApp and a Forensic Server. SynApp summarizes network events and keeps them temporarily. The forensic server manages a set of SynApps for a domain. It processes the queries received from outside the domain in cooperation with SynApps and sends back the query results. ForNet can identify some Network events, such as port scanning and TCP connection establishing, and tracks others using bloom filters. The architecture proposed in this thesis can be adjusted to collect the same data similar to ForNet but agentlessley without the need to employ SynApps.

Wei [24] proposed a model, based on client-server architecture, of network forensic system. The model captures network traffic, uses some adaptive filter rules to dump the malicious packets, transforms the packets into database values and mines the database. The data from IDS, firewall, remote traffic and Honeynet are also integrated by the distributed agents. The model analyzes the overall database and replays the misbehavior and can identify the attacker profile. Our proposed architecture can replace the agents used to collect the data from IDS, firewall, remote traffic and Honeynet, in addition to capturing the network traffic.

A simple framework was proposed by Tang et al. [27] for distributed forensics. It aims at providing mechanisms for collecting evidence, efficiently storing and analyzing forensic information through the use of distributed techniques. The model architecture involves agent and proxy. The data are collected, stored, processed, and analyzed by the agents. The proxies are responsible for generating the attack attribution graph and performing stepping stone analysis. Our proposed framework can be used to perform the job of agents, and the proxies can then be integrated with the framework to get access to and process the collected data.

Nagesh [21] implemented a framework for distributed network forensics that uses mobile agents to automatically collect network data from heterogeneous systems. The network traffic logs are collected and analyzed by the agents, and the results are sent to the network forensics-agent hosted on a server. The results are displayed on a user interface which enables the analysts to analyze the displayed network events and

specify the data to be collected as well. The framework implementation is scalable, provides real-time monitoring, and addresses the single-point of failure. Our proposed framework can be deployed on the server running the network forensic-agent. The framework will replace the agents to perform collection and analysis of the network traffic logs.

A dynamical network forensic framework (DNF) was developed by Wang et. al [28] that collects and stores data logs simultaneously at real-time, collects evidence and responds quickly to network attacks. The model is based on the multi-agent and artificial immune theories, and its architecture involves three agents in addition to a forensic server. The detector-agent works on capturing network traffic, comparing the captured data with intrusion behavior for match and then sending requests to the forensic-agent. The evidence is then collected by the forensic-agent and sent to the forensic-server to be analyzed and then replays the attack procedure. The architecture of above framework can be considered as agentless if the three agents are used on the network and not installed on each network endpoint. Our proposed framework can perform the job of the detector-agent while the forensic-agent and incident-agent can be integrated within the framework as security models.

2.2.0.2 Monitoring Tools

Monitoring IT infrastructure involves gathering data items from the monitored components at the hardware, service and application levels. The monitoring can be for the purpose of performance or availability, and some of the tools support security monitoring. Here, we review the most common and open source monitoring tools for IT infrastructure.

Nagios [3] is an open source tool for performance monitoring of IT infrastructure that involves hosts, services and network devices. Nagios supports plug-ins that are developed to overcome its limitations, such as support of virtual environments, and it has an active support community.

Zabbix [10] is another open source tool that supports large scale environments and high-performance data gathering. In addition, it can be used for performance and availability monitoring of servers, applications and network devices, and it is well supported by an active community.

Hyperic [1] is a monitoring software optimized for virtual environments. It has both an open source and paid versions, and it can automatically discover and monitor

software and network resources.

SolarWinds [8] is another monitoring tool that has a great community support. It is available as software as a service and self-hosted, and it provides VM support. It supports the management of and monitoring of systems, network and databases, in addition to IT security.

Although most of the data gathering tasks in these tools are performed using agents, some of the aforementioned tools support agentless data collection for some performance metrics, such as Nagios that supports partial servers monitoring agentlessly, and Zabbix. Moreover, all the tools are designed for performance and availability monitoring except SolarWinds that provides security monitoring using agents.

2.3 Agent or Agentless

The topic of agent-based vs. agentless monitoring has emerged front-and-center and been addressed by a number of online articles, blog posts and vendor white papers. However, most of those articles and white papers are not based on research studies and are often inaccurate, biased, incomplete or combination thereof. In addition, they lack the results of large-scale engineering or scientific studies. In this section, we review the articles that address agentless monitoring and compare it to agent-based monitoring.

In his article, Ingess [19] explained how agentless security will shape the future of cloud protection. The article listed several benefits behind the use of agentless protection that involve increased flexibility, seamless single interface management, IT cost savings and advanced malware and virus protection, to name a few. In addition, the author listed six elements that should be present in any agentless security solution: integrity monitoring, anti-malware protection, intrusion detection and prevention, firewall, log inspection and web reputation management. Our proposed agentless architecture provides a mechanism for collecting data that can be used for achieving the aforementioned six elements.

Caitlin [22] addressed the issue of deploying traditional security mechanisms with the emerged virtualized solutions rather than inventing or finding an architecture for agentless security protection. He claims that one of the key benefits of agentless security is that it does not intrude on the hypervisor level; it is done through virtual appliances. The article also discussed a set of goals for businesses that can be achieved by agentless security software, such as ease of administration, performance, no need

for updates, no need to maintain pattern files and little management overhead.

Ingmar [20] discussed about which monitoring method should be considered ultimately the best; whether agent-based or agent-less monitoring. The author claims that it is important to first identify what is being monitored in order to decide which method is better, as agent-based would excel with monitoring that does not require collection of large amount of data, while agent-less would succeed for deploying a full-scale monitoring solution which is not possible with agents. Moreover, the author conducted a comparison between agent-based and agentless monitoring based on several factors involving resource utilization and performance, stability and reliability, deployment, dependencies, security, and scope and functionality. However, the conducted comparison provides thoughts and opinions and is not supported by any studies or experiments.

In the current thesis, we propose an agentless architecture and develop a proof of concept showing that agentless monitoring is able to address several challenges of agent-based monitoring. Moreover, the proposed framework can be used as a basis for establishing large-scale studies to enrich the existing body of knowledge about the applicability and efficiency of agentless architectures and to answer several questions that are under discussion by professionals and researchers in this field, such as those presented in the cited articles.

Chapter 3

Framework Architecture

This chapter presents the implementation details of the proposed framework. First, the general architecture of the framework is introduced. Then it proceeds with discussing the development details subsequently in the following sections.

3.1 Framework Design

Before starting with the design details of the proposed framework, we need to specify the requirements of this framework. Following is a summary list of the framework main requirements:

1. The need is for a security monitoring framework that can be used for monitoring endpoint devices while eliminating the need for installing monitoring agents on the hosts being monitored.
2. The framework should provide a level of security to protect against different kinds of possible security attacks.
3. The framework should be scalable in a way that enables the monitoring of large number of endpoints while maintaining the scanning performance.
4. The framework should perform the data retrieval with low network latency so that it can be used to scan large number of hosts with no impact on the network performance.

5. The framework should be designed to involve flexible, and easy-to-use components.

The design phase will state the architecture of the framework and the main components constituting the framework based on the specified requirements. Finally, the development phase of the framework will be carried out.

Figure 3.1 depicts the framework main components and how they communicate with each other. A brief description of each component is provided as follows.

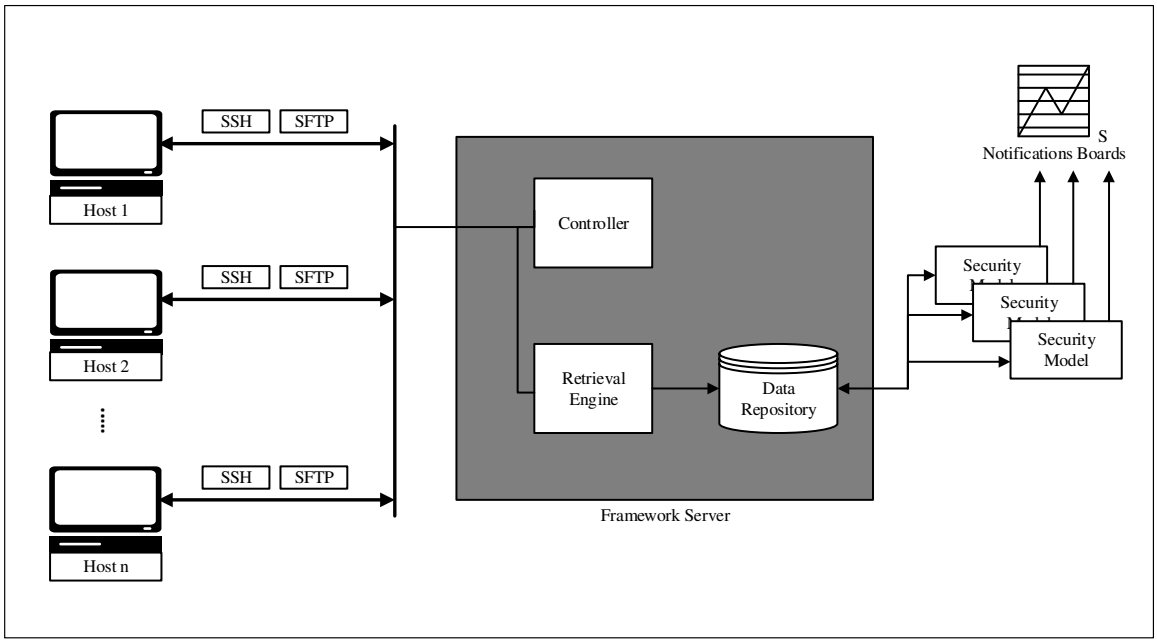

Figure 3.1: Framework architecture

- **Controller**

  The controller is a management service that manages and establishes connections with the monitored hosts on a regular basis or on request. In addition, it is necessary to make sure that the connections and the data gathering are performed in a secure way to protect against attacks targeting data confidentiality, integrity, and availability. For this purpose, the controller will be responsible for handling the mentioned operations securely. The appropriate security methods, that will be adopted, will be specified and discussed later.

- **Retrieval engine**

  The Retrieval engine is the main component of the framework in charge of col-

lecting host-based data from the monitored hosts. The framework is supposed to provide a configuration means, embedded within this engine, enabling the security officers or the framework administrators to specify the list of hosts to be monitored, in addition to determining the different host-based data items that should be retrieved from each monitored host. This engine is not supposed to establish direct connections with the monitored hosts. Scanning requests should be forwarded to the controller that will establish the connections accordingly in a secure way.

- **Data repository**
  The collected data from the monitored hosts are stored in a storage media (e.g., log files, light database, etc.) in a special format. This can be useful for several reasons. First, this repository will work as a shared repository that can be accessed by various security scanning modules, integrated with the framework, at the same time in one-to-many relationship, which may enhance the performance and the accuracy of the monitoring process. Second, different reports can be generated from the collected data on a historical basis.

- **Security models**
  The main purpose of the proposed framework is the monitoring of the target computing hosts. As mentioned in the previous sections, the framework components will work on collecting various host-based data items that will be used as a feed to the security models which are responsible for distinguishing normal behavior from suspicious and malicious ones. In this work, we are not going to propose any security models and the framework can be integrated with existing models targeting various types of malicious activities.

- **Notification board**
  The results of the security checks done by the security scanners integrated with the framework can be displayed to the operators or security officers via the notification boards, particularly the critical alarms and a summary of the last scanning routines.

- **Employed protocols**
  For the sake of interoperability, the framework is implemented in such way that it can be used to monitor different computing hosts running on different platforms (e.g., Linux, Windows, etc.). The framework connects with the monitored

hosts and collects data items by using standard protocols, such as SSH, SFTP, and so on, that are supported by most operating systems.

The suggested work flow of the proposed framework is shown in Figure 3.2. The first step "**Identify the host IP to scan**" decides which host or group of hosts will be scanned at the current timestamp. This can be modeled using different selection criteria including priority, last scan time, etc. The second step "**Determine the data items to collect**" works on selecting the different host-based data items that will be collected for the hosts identified in Step 1 and the related scripts/code that will be executed on the target hosts to collect the specified data items. Steps 1 and 2 are performed by the Retrieval engine. In step 3 "**Establish secure connection,**" the controller will receive a request with the target host information. Based on this request, the controller will establish a secure connection with the specified host and informs the retrieval engine upon success which will accordingly send subsequent requests representing the data items to be collected. The controller will execute the corresponding commands/scripts to collect the required host-based data items as in step 4 "**Collect data items**." In the last step 5 "**Store retrieved data**," the collected data will be stored in the data repository. This process is repeatedly executed on regular basis to keep the endpoints monitored up to the moment.

3.2 Framework Implementation

According to the framework architecture depicted in Figure 3.1, the framework consists of a set of core components which handle raw data collection and processing, and another set of components that manage the collection process. This section will present the technical details of all framework components and how they are implemented.

3.2.1 Collection process management

The framework provides a couple of configuration files which enable the security officers or the framework administrators to maintain various framework settings. The configuration files provide a centralized mechanism for setting up the information needed by different components of the framework at once.

The framework relies on the following main configuration files:

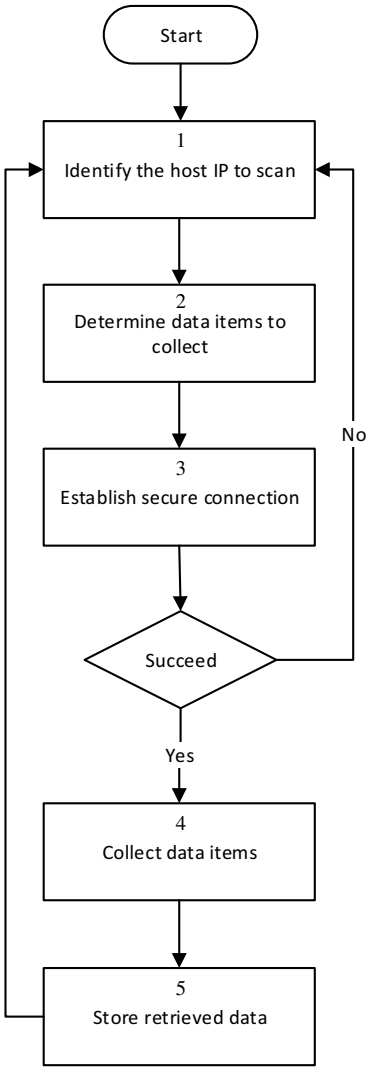

Figure 3.2: Work flow

- **main.conf**
  This file is used to configure general settings for the framework such as the authentication method, the time gap between each two subsequent collections, the framework log/debug files, etc. The file structure of *main.conf* is shown in Figure 3.3.

- **hosts.conf**
  This configuration file enables specifying the list of hosts or endpoints to be monitored. In addition, it lets us disable the monitoring of any host without the need to remove the host information from this file. In this way, any excluded

```
[main]
#Specify the method of authentication that will be used while
#establishing connections with the monitored hosts
#[1=password authentication, 2=public key authentication].
auth_method=2

#Specify the time period in seconds between each two subsequent
#raw data collection processes for each monitored host.
scans_schedule=30

#Specify the full path of the file that will be used by the
#framework for logging information and traces related to the
#functionality of the framework.
logs_file=framework.log
```

Figure 3.3: Main configuration file structure

host can be remonitored easily just by enabling the corresponding monitoring option in this file. The structure of this configuration file is depicted in Figure 3.4.

```
[host1]
#Specify the name of the host being monitored. It is recommended to use
#the actual name given to this host.
name=fileserver

#Specify the IP address of this host. Make sure that the IP address is valid
#and that it is reached from the server on which the framework is running.
#If not, the monitoring process of this host will fail and the host may get
#compromised
ip=192.168.50.15

#Specify the username that can be used to access this host in order to
#collect raw data.
user=root

#Specify the password, of the username stated above, that can be used to access
#this host in order to collect raw data. This is not required if the login to hosts
#is enabled via public key authentication.
passwd=my secret password

#Disable or enable the monitoring of this host[yes/no]
enabled=yes

[host2]
name=Asem_terminal
ip=192.168.50.1
user=test
passwd=test
enabled=no
```

Figure 3.4: Hosts configuration file structure

- **rawdata.conf**

  All the raw data that should be collected for each endpoint, during the monitoring process, are specified in this file. Moreover, this file enables framework administrators to handle the way in which each raw data is collected without the need to modify the framework source code. You only need to state the name of the script file containing the command or group of commands that will handle the raw data collection. Figure 3.5 depicts the structure of the raw data configuration file.

```
[data1]
#Specify the name of the data being collected. This assigned name will be stored
#along with the data in the data repository.
name=logs

#Specify a unique numerical id for the data being collected. This will be
#considered as a unique identifier to enhance the performance of referencing
#the data in the data repository, for example.
id=1

#Specify the name of the script containing the command or group of commands
#that will be executed in order to collect this data.
command=logs

#Disable or enable the data collection and retrieval[yes/no]
enabled=yes

[data2]
name=active-processes
id=2
command=actvproc
enabled=yes

[data3]
name=open_connections
id=3
command=open_conns
enabled=no
```

Figure 3.5: Raw data configuration file structure

3.2.2 Data collection

The data collection or retrieval process goes through three main steps. First, a secure connection is established with every host under monitoring by the framework. Then the actual retrieval of the raw data starts. Finally, the retrieved data are stored in a storage media and shared with the security scanners integrated with the framework.

3.2.2.1 Endpoints authentication

One role of the controller is establishing secure connections with the monitored hosts to be able to start the raw data collection process. To establish a connection with any host, the controller must go through an authentication procedure. The framework supports two methods of authentication, namely, password authentication, and public key authentication. The framework administrators can choose which method to be used for the authentication by setting the *auth_method* in the framework configuration file *main.conf*.

- **Password authentication**
  Password authentication requires specifying the username and password for each host in the *hosts.conf* file. The username and password will be used as credentials for establishing connections with the monitored hosts. The framework supports this method in order to increase usability and it should be used while restricting access to the *hosts.conf* file.

- **Public key authentication**
  Public key authentication provides a cryptographic mechanism for performing secure and passwordless authentication [4]. In addition to its security, it facilitates the implementation of single sign-on across the monitored hosts. In this method, a key pair is created for each user that will be used for establishing connections with each host. The private key stays on the framework server while the public key is sent to the monitored host using a specific utility. The monitored host now will allow access for the framework which has the corresponding private key. The process of creating a key pair must be done only once for each monitored host at the beginning when the host is added to the list of hosts to be monitored by the framework.

3.2.2.2 Raw data collection

One of the contributions in this work is to come up with a flexible and generic framework that can be customized to satisfy the needs of the organization or the company deploying the framework without modifying the source code of the framework. Therefore, the retrieval engine will be implemented to handle the general process of the raw data collection while enabling the end users to write their own code in charge of collecting a new raw data item of interest and integrate the written scripts easily with

the framework to be used for collecting the new raw data along with the existing ones.

The general process of raw data collection is depicted in Figure 3.6. The data collection process for a host starts when the controller establishes a connection with the targeted host successfully. The data retrieval engine reads the corresponding code files/scripts responsible for collecting the raw data and checks their validity. Then the code is sent via requests to the controller. The controller gets the received code executed on the targeted host and then retrieves the collected data and sends it back to the data retrieval engine. Finally, the data retrieval engine formats the retrieved data in a specific structure and stores it in json files.

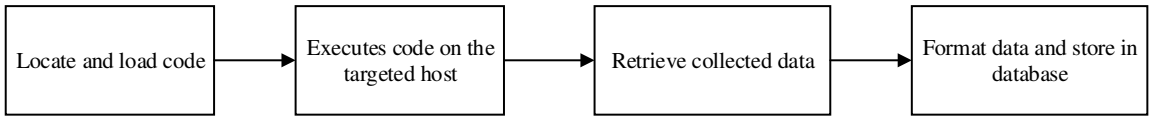

Figure 3.6: Data collection procedure

The core functionality of the framework is shown in Algorithm 2.

Algorithm 1 Framework main algorithm

```
1: procedure MAIN
2:     hosts ← load(hosts.conf)
3:     rawdata ← load(rawdata.conf)
4:     main_config ← load(main.conf)
5:     for config in read(main_config) do
6:         Initialize the framework global settings
7:     end for
8:     while True do                                   ▷ framework main loop
9:         Check system resources
10:        if Available memory ≤ memThreshold or disk space ≤ spThreshold then
11:            Warning
12:        end if
13:        for host in read(hosts) do  ▷ Hosts scanning begin  ▷ Execute in parallel
14:            Exclude hosts for which monitoring is disabled
15:            if auth_method ==1 then
16:                Establish connection using password authentication
17:            else if auth_method ==2 then
18:                Establish connection using public key authentication
19:            else
20:                Error
21:                End current host scan
22:            end if
23:            for data in read(rawdata) do
24:                Exclude data with disabled status
25:                loc ← locate_code(data)
26:                code ← load(loc)
27:                validate(code)
28:                buf ← execute(code)          ▷ execute code on the monitored host
29:                tmp ← retrieve(buf)
30:                if size(tmp) exceeds maximum host quota then
31:                    Block the monitored host and report as compromised
32:                end if
33:                formated_data ← format(tmp)
34:                store(formated_data)
35:            end for
36:        end for
37:    end while
38: end procedure
```

3.2.2.3 Light agent mode

In some cases, the data collected through the agentless architecture does not fulfil the needs of anomaly detection systems due to the need to install some tools on the monitored hosts to collect data that can not be collected remotely. In this research, the term "light agent architecture" is used to reference those cases when the agentless architecture is used to collect most of the data required to perform anomaly detection while installing some light tools or agents on the monitored hosts to collect other kinds of data. As detailed in the previous sections, the framework mainly depends on employing built-in utilities and commands to collect raw data. The data that can be collected via the agentless architecture will vary through the different platforms. In addition, the framework enables the security administrators to write their own scripts for collecting data. Therefore, advanced scripts may be used to collect data when built-in utilities do not exists. However, some security administrators would prefer to install some tools on the monitored hosts instead, and this is what we call light agent architecture. In this research, our focus is on the fully agentless mode and what can be collected through the agentless architecture.

3.2.2.4 Data storing

As mentioned in the design phase of the framework, the data retrieved from the monitored hosts can be stored in a storage media (e.g., log files, light database, etc.). A database such as MongoDB [2] can be used for this purpose. However, in our implementation of the framework prototype, we will use JSON files for storing the collected data from the monitored hosts. JSON files will be used for the fact that it allows storing data in a standard data interchange format which is lightweight. Moreover, JSON is considered a language-independent format, being supported by many different programming APIs. This will facilitate the data interchange with the various security models integrated with the framework. Furthermore, the lightweight feature of JSON files will have a good impact on the framework performance and will reduce the overhead on the storage disks.

For flexibility, the framework administrator has the ability to configure a specific directory that will be used by the framework as a central location for creating json files and storing the retrieved data. This can be achieved by setting *database_dir*, in *main.conf*, to the path of this directory. A separate json file will be created for each scan of every computing host. The pattern *hostname_yyyy-mm-dd-hh:min.json*

is used for naming the created json files. The framework checks regularly the free space of the disk where the json files are stored and warning alerts are generated when the free space is less than some threshold (set to 4 GB in our implementation).

3.2.3 Collected raw data

This section explores which type of data can be collected through the framework architecture. No much focus will be paid at this stage on how the data will be processed on the framework side. We would explore, in particular, what can be collected related to the following types of data:

1. **User activity data**
   Following is a list of user activity data that can be collected through the framework architecture.

   - Last logon, either for local users or domain users
   - Failed login attempts. By default, Windows platform does not provide such information and account auditing need to be enabled by the framework administrator on the monitored hosts.
   - User password last set
   - Group memberships for a user
   - List of users logged on and the session information for each user
   - List of active sessions
   - Windows event logs can be collected, and different user activity data can be extracted from these log files.

2. **Process table**
   Regarding processes running on the monitored hosts, the framework can collect the following data:

   - List of all processes running on the system, the extracted information for each process is depicted in the following tuple:

     [*PID, Session#, Session name, Image name, Memory usage*]

- List of processes using memory space greater than certain value
- A history of all processes running (successful) or tried to run (failed) on the system. By default, Windows platform does not provide such information and process tracking events need to be enabled by the framework administrator on the monitored hosts.

3. **File system**
   Operations on the file system of a host can play an important role in building malware detection models, for example. In the current version, the proposed framework is able to collect the following data:

   - The set of all files modified within a specific period of time
   - The set of all files created within a specific period of time
   - To collect data related to the system calls, the framework may need to run some utilities or advanced code scripts on the monitored hosts that work on intercepting I/O requests.

4. **Registry**
   For operations on Windows registry, the framework is able to extract the following data:

   - Values of specific registry keys
   - Export of the whole registry keys
   - Registry operations, such as recently added keys, modified keys, etc., can be figured out indirectly by measuring deviation between two subsequent reads.

5. **Network connections**
   The network connections data that can be gathered by the framework are summarized as follows:

   - The framework can extract data from the DNS resolver cache of a monitored host which stores the IP addresses for the websites recently visited from the host.
   - Extracting data related to network connections established to and from each monitored host requires scanning of the inbound and outbound network traffic. The proposed framework is agentless and does not have access

to the network interfaces of the monitored hosts. However, one workaround solution may be through routing specific packets from the monitored hosts to the server running the framework.

6. **Resource consumption data**
   The following resource consumption data can be collected by the proposed framework:

   - Available physical/virtual memory
   - Total physical/virtual memory
   - Disk usage
   - CPU usage

3.3 Framework Generality

Beginning at the early stages of this work when the idea of building an agentless framework for endpoints monitoring started to form, the generality of the framework was the core of our brain storming. As any software framework, the proposed framework should provide a generic functionality that can be extended by additional user-written code [7]. According to Pree [23], the software framework should consist of both frozen and hot spots, where the frozen spots define the main components of the framework or its overall architecture and how they are connected with each other. While the hot spots define the other parts of the framework that enable the end-users or programmers to extend the functionality of the framework by writing their own code, for example.

3.3.0.1 Generality as a software framework architecture

Now, and as we reached the end of this chapter, in which we presented the framework architecture and the details of designing and implementing the proposed architecture, we will discuss how the proposed framework architecture fits within the general definitions of software frameworks. By having a look again at the proposed framework architecture depicted in Figure 3.1, the work flow of the framework shown in Figure 3.2, and the way in which those components are integrated, it would be easy to recognize the frozen and the hot spots of the introduced framework. The framework

consists of those core components (e.g., controller, retrieval engine, etc.) that manage the data collection process, which represent the frozen spots. The end-users do not need to change the code of those components in order to add extra functionality. On the other hand, the framework enables the users of the framework to configure the functionality of the framework the way that fits the requirements of their corporate environments. The framework provides a set of configuration files through which its functionality can be maintained or extended. The types of raw data that will be collected by the framework are not fixed or programmed during the framework implementation. The end users can write their own code and scripts to collect specific data, and the framework would manage the data collection process using the code and scripts written by the framework users. This represents the hot spots of the framework.

3.3.0.2 Generality in the framework use cases

As we can observe from the literature review in Chapter 2, almost all the existing related works are not generic and the proposed approaches are designed and implemented with a focus on specific types of data. However, our proposed framework is generic, and it is not designed and implemented to collect only limited kinds of raw data. This section will address the first research question of the current thesis.

RQ1 : What is the scope of endpoints security monitoring that can be performed using the proposed generic agentless architecture?

As mentioned in the collected raw data section 3.2.3, the framework can collect data related to memory, file system, resource usage, processes, network connections, network traffic, Windows registry, and other parts of the computing hosts that affect the behavior of the computing hosts. Framework users or administrators can use it to collect a wide set of data, and they can integrate their extensions very easily through the configuration files of the framework. Moreover, the collected data from different sources could be combined together for performing more sophisticated security monitoring. This variety of the data items that can be collected by the framework enables it to be used for scanning and detecting wide range of host-based attacks.

As the framework can be used to collect such wide range of raw data, the use of the framework is not limited to specific use cases. For example, the data collected by the framework might be used to build a security defense model that detects malware based on the data captured from the host memory. Another use case could be building

a model that detects ransomware based on the behavior of the file system. A third use case could target the detection of viruses based on the data related to the events of the file system, registry and running processes. Moreover, Chapter 5 of this thesis will present a use case about building a real-time intrusion detection for detecting malicious break-in attempts and masqueraders based on the data collected from the system logs of the failed and successful logins. Those are just few examples to show how the framework can be used in various use cases.

3.4 Summary

In this chapter, we have proposed an agentless architecture for monitoring endpoint computing hosts remotely. The next chapter will evaluate the security and the efficacy of the proposed agentless architecture.

Chapter 4

Evaluation and Analysis

This chapter provides an evaluation of the framework. In the first section, the security properties of the framework are being evaluated. Then the second section provides an evaluation of the framework performance.

4.1 Framework Security Evaluation

In this section, the security assumptions of the framework will be presented. Then we will study and examine the security of the proposed framework.

4.1.1 Framework functionality

Before starting the security evaluation process of the framework, we need to understand how the framework is supposed to work and what is needed for the framework to work properly.

- The framework is installed to run on a server connected to the network on which the endpoints to be monitored are running.
- The framework has access to the endpoints it is supposed to monitor.
- The server on which the framework is running is considered to be secure, to prevent against various attackers' attempts of manipulating the framework configurations and tampering the collected data.
- The security manager or administrator of the framework is supposed to configure the framework properly by setting the right values for all parameters in the

framework configuration files, namely, *main.conf*, *hosts.conf*, and *rawdata.conf*.

- Access to the configuration file *hosts.conf* has to be restricted.
- In case of using "public key authentication" to establish connections with the monitored hosts, the security manager of the framework is supposed to send the public key of each monitored host to the monitored host using the proper utility.
- The security manager of the framework is supposed to add the required IP address of each new host to the configuration file *hosts.conf*, and maintain this file up to date thoroughly. However, the framework can be easily updated to detect the new hosts connected to the network.

4.1.2 Attacker model

Following the Dolev-Yao threat model [16], any violation of the framework operation is considered as an attack. The attacker is able to intercept, overhear, listen, and modify data exchanged between the framework server and the endpoints. The power of the attacker is only limited by restrictions imposed by the employed standard cryptographic protocols, and the security mechanisms adopted on the framework server. The assumptions considered for the attacker model are as follows:

- The attacker can sniff the network traffic between the framework server and each monitored host.
- The attacker can intercept and manipulate the network packets exchanged between the framework server and each monitored host.
- The attacker can capture the authentication related network traffic and the network traffic of the data collection requests sent from the framework server to each monitored host, and record the corresponding data for later use. For example, to conduct replay attacks.
- A monitored host may get compromised and controlled by the attacker.
- The attacker can spoof the IP address of any of the monitored hosts.
- The attacker can send connection requests to the framework server.

4.1.3 Informal security analysis

Having set up all the machinery of the framework as stated in Section 4.1.1, and given the attacker model described in Section 4.1.2, this section discusses potential attacks that could violate the security properties of the framework, and how the proposed framework design protects against such attacks.

4.1.3.1 Credentials sniffing attack

Because the attacker has the capability to eavesdrop on the network traffic established during the authentication with any monitored host, he may learn the credentials of the hosts, for example, and then use them to get control over the monitored hosts. However, the proposed framework protects against this threat by, first, using standard and secure protocols to establish connections in a secure way (e.g., SSH). Second, The framework provides another level of security against this attack by using "public key authentication" to establish connections.

4.1.3.2 Replay attacks

The attacker may use the recorded packets to do several actions, such as establishing connections with the monitored hosts, gathering various data, disturbing the functionality of the targeted hosts, etc. However, the framework depends on using standard and secure protocols (e.g., SSH) that are proved to protect against replay attacks in general.

4.1.3.3 Spoofing attack

As discussed in the attacker model, the attacker can spoof the IP address of any monitored host. In this case, the framework will not be able to establish connection with the attacker host as the attacker has to create a user account with the same credentials used by the host with the spoofed IP. In addition, the framework raises an alert when it fails to establish connection with any host. Even in the case where the attacker was able to guess the credentials, the attacker will not be able to do any malicious activity against the framework.

4.1.3.4 Man-in-the-Middle attack

If the attacker was able to inject himself between the framework and a monitored host, he will be able to either sniff the exchanged traffic and hence he will learn nothing from the encrypted traffic, or he will try to tamper the transferred data and in this case the tampered packets will be discarded. The attacker may use this attack to prevent monitoring of a specific host by keeping tampering the collected data sent back to the framework; however, the framework raises an alert if no data are received from any host for a predefined period of time.

4.1.3.5 DoS attacks

DoS attacks can be launched by the attacker against the server running the framework to impact the functionality of the service provided by the framework. However, the proposed framework enables the framework administrators to set up thresholds for the minimum required memory and disk space, and alarms will be raised if the available memory or disk space go below those specified thresholds. In addition, the future work can adopt a model that works on detecting various attacks targeting the availability of the framework.

4.1.3.6 Compromising a monitored host

If any of the monitored endpoints got compromised by the attacker and became under his control, the attacker might try to manipulate the source raw data before being collected to mimic a normal behavior and avoids being detected. However, it would be difficult for the attacker to manipulate all the data, as the framework works on collecting several sets of data, most of them are generated and manipulated only by the operating system. In addition, the data are collected on a regular basis and stored in a database, and any deviation from the previous records of the data will be detected by the security model.

4.2 Framework Efficiency Evaluation

As mentioned in Chapter 3, one of the requirements of the proposed framework is to be scalable in a way that enables the monitoring of large number of endpoints while maintaining the scanning performance. In this section, the efficiency of the framework will be examined in terms of resource consumption. To evaluate the performance of

the proposed framework, we have run several experiments where the framework has been used to monitor a group of hosts while measuring the overhead on the framework server in terms of memory and time required to collect data.

4.2.1 Experimental Setup

The network topology of the testbed used to run the experiments is depicted in Figure 4.1. The framework has been installed and configured on the machine named "Framework Server." To evaluate the scalability and the performance of the framework, the framework should be used to monitor a large group of computing hosts. However, preparing such testbed is costly and time consuming. As an alternative, we have used the framework to monitor one physical device and 11 virtual machines. All the physical and virtual machines are configured to run on the same LAN network along with the framework server. The physical host and the virtual machine instances run Ubuntu 16.04 LTS. Finally, the framework has been configured to monitor all those machines.

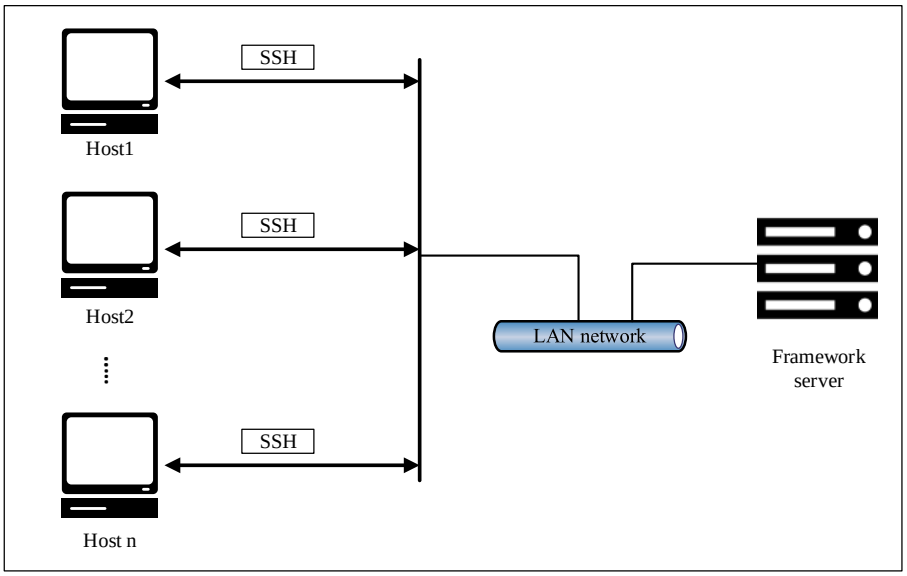

Figure 4.1: Performance evaluation testbed

The specifications of the framework server and the monitored physical and virtual hosts are shown in Table 4.1. The table states the memory and the platform of each host.

Table 4.1: Hardware and platform specifications

Given Name	Processor Type	Memory	OStype	Platform
Framework server	Core i7- 2.70GHz	4GB	64-bit	Win7
Physical host	Core i7- 2.80GHz x 8	8GB	64-bit	ubuntu 16.04 LTS
Each virtual host	–	1GB	64-bit	ubuntu 16.04 LTS

4.2.2 Collected data

In our experiments to evaluate the efficiency of the framework, the framework has been used to collect the following data from the monitored hosts:

- Successful login attempts made by users and the currently logged-in users.
- Failed login attempts.
- List of all processes running on the system.
- Performance related data of all running processes in the system.
- Available physical/virtual memory.
- Total physical/virtual memory.
- Disk usage.
- CPU usage.

4.2.3 Evaluation

We started our experiments by running the framework to monitor one instance, and then measured the resources needed (memory) to run the framework and the time it takes to collect data from the monitored instance. We repeated this experiment five times and then calculated the average for the consumed memory and the elapsed time. After that, we kept adding one more instance to the list of the monitored instances and measuring the memory overhead and the elapsed time. We limited the number of monitored instances to 12 due to the resources needed to run extra virtual instances. The results are presented in Table 4.2.

Table 4.2: Overhead in terms of memory and data collection time

Number of Monitored Instances	RAM(K)	Data Collection Time(sec)
1	12,912	0.518
2	13,060	0.61
3	13,068	0.619
4	13,060	0.653
5	13,060	0.667
6	13,072	0.67
7	13,072	0.690
8	13,132	0.688
9	13,184	0.689
10	13,128	0.701
11	13,304	0.704
12	13,312	0.701

Figure 4.2 shows the average time (in seconds) it takes to collect the data from each instance and store the collected data in the corresponding database. The framework took about half second to collect the mentioned data from an instance when it was used to monitor just one instance. To test the scalability of the framework, we kept increasing the number of the monitored instances to measure the effect on the performance of the framework. We can see that the average time needed to collect data from each instance was not impacted significantly by increasing the load on the framework by adding more instances to be monitored.

Figure 4.3 depicts the amount of memory consumed by the framework on the framework server. The framework needed just about 12 MB of memory. There was only a slight change in the amount of memory consumed by the framework when the number of monitored instances was increased.

4.3 Discussion

As we are done with performing our evaluation of the proposed framework, in this section, we will address the second research question of this thesis.

RQ2: How can agentless architecture address the scalablity, performance and security issues of the agent-based architectures?

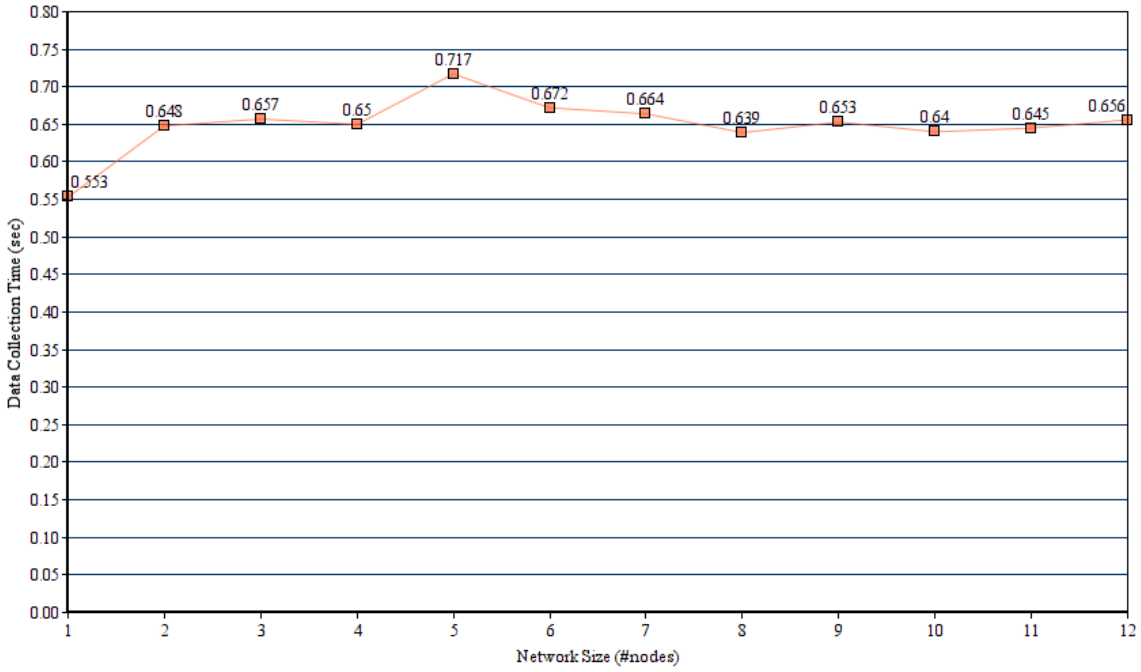

Figure 4.2: Data collection time per instance

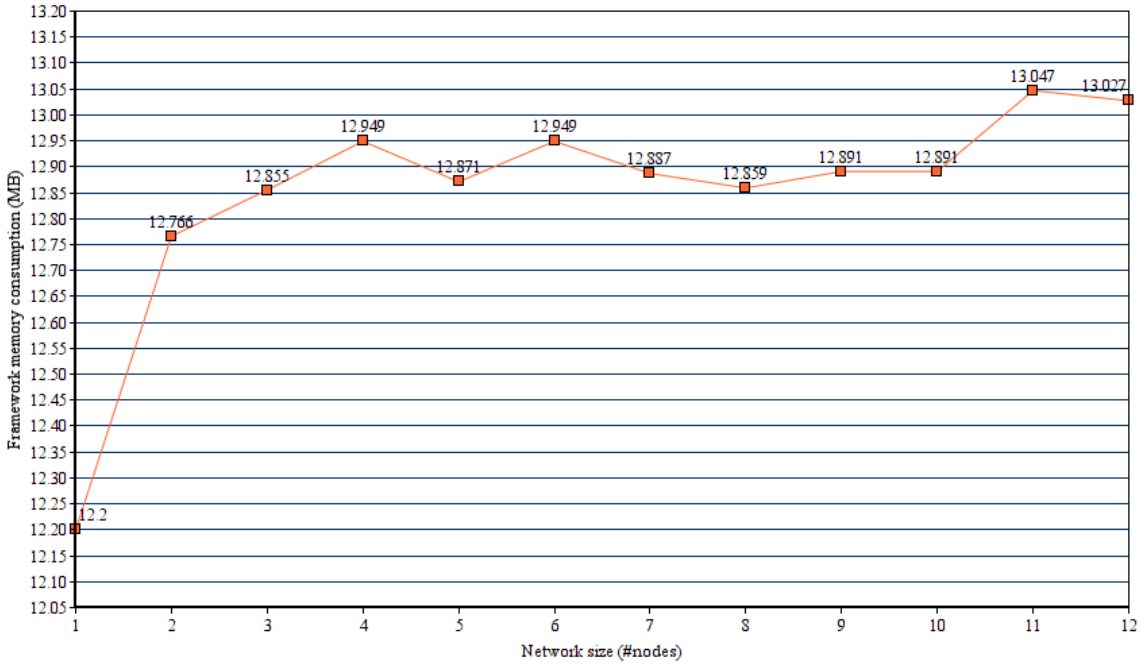

Figure 4.3: Framework memory overhead

We have observed from the performed security evaluation the feasibility of designing and building agentless architecture for performing endpoints security monitoring

in a secure way. Besides the presented discussion of the ability of agentless architecture to protect against various kinds of attacks, we need to highlight how the agentless architecture addresses the other security drawbacks of agent-based architectures. One of the major drawbacks is that agents can provide extended attack surface for targeting monitored endpoints. The elimination of installing agents on endpoints guarantees the elimination of such attack surface provided by agents. Moreover, agentless architecture provides chance for companies to be responsible of their monitoring infrastructure and reduce the dependence on third-parties. In this case, even if the core part of the agentless architecture got targeted by attacks, the impact of those attacks would be limited as long as the collected data are locally controlled and monitored.

With regard to the capability of agentless architecture to be scalable and monitor large number of endpoints without degrading the level of security provided for the monitored endpoints. Our evaluation experiments showed that increasing the number of hosts, in our proposed architecture, has a very slight impact on the time needed to collect data from the monitored endpoints to perform security monitoring by the integrated security models. In addition, the time taken to collect data was below one second in the worst case which highlights the good performance that agentless architecture has to perform security monitoring.

However, the evaluation experiments performed in this thesis have some limitations due to the cost and time of performing evaluation in large networked environments. To derive a general conclusion about the scalabilty and performance of agentless architectures, there is a need to perform a large-scale evaluation in large networked environments, and this would be part of the future work that can be performed using our proposed framework.

4.4 Summary

We have performed, in this chapter, security analysis to evaluate the security of the framework against various attacks. In addition, several experiments have been conducted to evaluate the efficacy of the framework. The framework has been shown to be secure against potential attacks, and results showed that the framework had stable performance despite how many instances are monitored. The next chapter will present a real-world use case of the proposed framework.

Chapter 5

A Real-time Intrusion Detection Use Case

This chapter presents a use case that shows how the proposed framework can be used to build a real-time intrusion detection model that focuses on detecting abnormal behaviors based on the data collected using the framework.

5.1 Intrusion Detection Model

In this use case, an intrusion detection model will be presented. The model is for detecting break-ins into computing systems by monitoring users' activities for abnormal patterns. We partially adopt the idea presented in [15]. In the presented model, the behavior of a given user with respect to a given account or machine is captured as an activity profile. The activity profile serves as a description or signature of normal activity for its particular user, account or machine. Statistical metrics and models are used to characterize the observed behavior. We will start by discussing the threat model. Following that, we will present the statistical metrics and models, and the activity profiles used in the intrusion detection model under consideration.

5.1.1 Threat model

The model is intended to detect the following types of threats.

5.1.1.1 Attempted break-in:

For an attacker to access an unauthorized account, the attacker needs to get hold of a valid password to that account. This may be conducted through some form of social engineering or phishing attack. But a common approach consists of conducting some dictionary attack by trying several candidate passwords. This may lead to unusually high level of unsuccessful login attempts to a single account.

5.1.1.2 Successful break-in or Masquerade attack:

A successful break-in by an intruder will result in this intruder masquerading (after, for instance, social engineering or dictionary attack) as a legitimate user. An assumption that needs to be looked at is the high possibility that the intruder might log into the account at a login time, or an IP address that is different from that of the legitimate user.

5.1.2 Metrics

In the proposed model, a metric is a representation of a random variable that models accumulated values of quantitative measure over a period of time. We consider the following two types of metrics proposed by Denning [15]:

- **Event counters:** number of events satisfying specific properties.
  Examples: number of successful logins over a time period, number of failed login attempts over a time period.

- **Interval timers:** duration between two related events.
  Examples: time interval between consecutive logins into an account; login interval.

5.1.3 Statistical models

To determine whether a new record of specific type of data recently collected using the framework is abnormal, a statistical model is used along with the archived records (past observations) of the same type of data stored in the framework database. The following models suggested by Denning [15] may be considered.

- **Mean and standard deviation model:** computes mean and standard deviation from past observations; if a new observation is not located inside a

confidence interval limited by $d \times std$ from the mean, it is considered abnormal. This model can be applied to both event counters and interval timers.

- **Time series model:** an event counter along with an interval timer are used with this model. The model considers the values, the order, and inter-arrival times of a sequence of collected records (observations) $x1, ...xn$ of a data x. If the probability of occurrence of a new collected record of data at that time is too low, it is considered abnormal.

More sophisticated models, e.g., using machine learning, could be considered; but this is out of the scope of the current thesis.

5.1.4 Profiles

We consider the following profiles as examples:

- **LoginFrequency:** can be represented using an event counter metric which measures the login frequency over a period of time. To capture variations in login behaviour, login occurrences can be modeled using an array of event counters labeled by time of day (hour vs. shift or evening vs. night), and day of week (work days vs. weekend). Unauthorized access to an account by masqueraders can be detected by monitoring login frequencies specifically when the access happens during off-hours in which the account is less likely used by the legitimate user.

- **LastLogin:** can be captured using an interval timer that measures the time elapsed since last login compared against some predefined threshold. Detecting break-in on a dead account, or account with limited activity (for a certain period of time) could be done with the help of this profile.

- **PasswordFails:** captured using an event counter that measures failed password attempts at login compared to some predefined threshold and time window (fairly short one). This can be monitored for individual accounts and for all accounts taken together; so two different thresholds can be defined. This could be useful for detecting break-in attempts.

- **SessionElapsedTime:** can be represented using standard deviation statistical model that measures the elapsed time per session. This model could be useful for detecting masqueraders.

5.1.5 Implementation

As a proof of concept, in our initial implementation, we will focus on a subset of the intrusion features as follows:

5.1.5.1 Failed login occurrences

Failed login occurrences over a time interval will lead to two types of events: a warning or an intrusion.

Algorithm 2 depicts the model that analyses the failed login occurrences of a given user with respect to a given account of machine and generates a warning or intrusion alert. This model takes as input a sequence of timestamps of consecutive failed login attempts and the timestamp of the following successful login, if any.

Algorithm 2 Failed login occurrences model

Input: failed login times, successful login time

Output: warning/intrusion

```
1: procedure
2:     Count number of consecutive failed logins over a time interval
3:     if number of failed logins is greater than threshold then
4:         generate warning
5:     end if
6:     if sequence of failed login is followed by successful login then
7:         flag the session as intrusion
8:     end if
9: end procedure
```

In case where the number of failed logins over a time interval is above a certain threshold, a warning will be generated indicating a password cracking attempt.

In case where a sequence of failed logins (above a certain threshold) is followed by a successful login over a time interval, a successful break-in through password cracking will be reported as an intrusion.

5.1.5.2 Model building

To build the model, we need first to understand the normal login behavior of a given account over a long period of time and capture that behavior using *PasswordFails*

profile as discussed in 5.1.4. Since we are interested here in analyzing the failed logins of a given account, the PasswordFails behavior would be characterized using a time series statistical model and event counter metric, where the number of failed logins for a given user is counted for a time window of 1 minute.

For this purpose, we have used the testbed depicted in Figure 5.1 to collect the required data. The test environment consists of a server running the framework and another Linux machine used frequently in a real work environment. The framework has been used to monitor the Linux machine for a period of three consecutive work days. The framework has been setup to collect every minute the number of failed login occurrences for a given account, from Linux logs, that happened during the last minute time window. The collected data for the whole period (three days) have been stored in the framework database. We would like to mention that the purpose of our conducted experiments is only providing a proof of concept for evaluating the framework use case. The main aim of the current thesis is proposing an agentless architecture, and follow-up studies will use the introduced framework to conduct large scale evaluation in our future work.

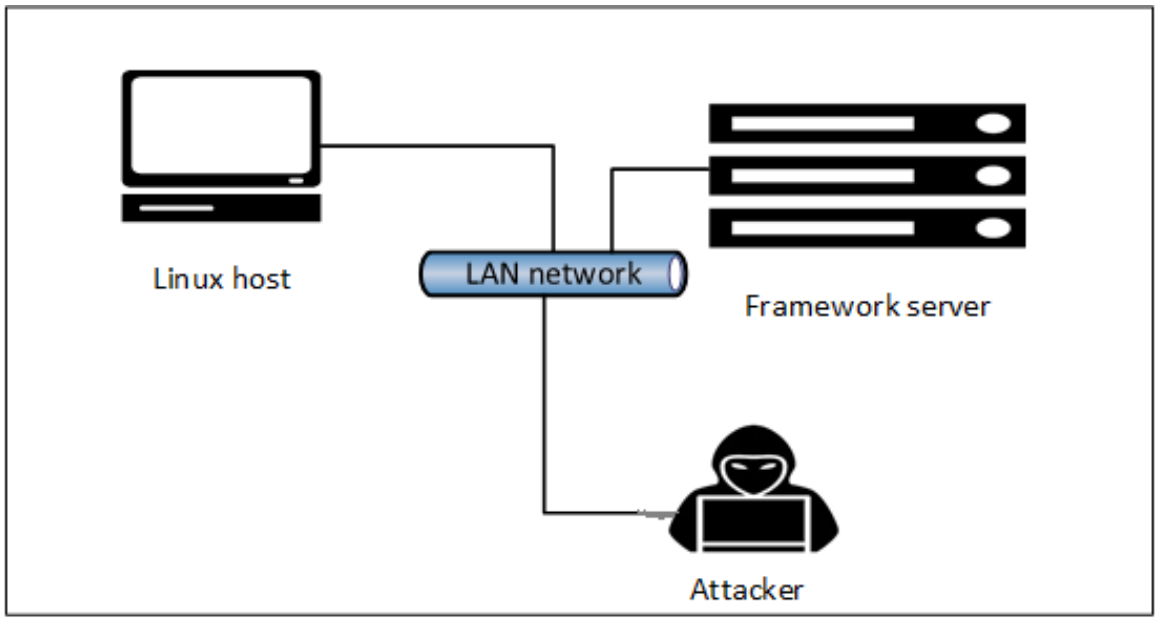

Figure 5.1: Intrusion detection model testbed

After the successful collection of the failed login attempts data, we have used the collected data to build the *PasswordFails* profile for such account that represents its normal behavior. A visualization of the profile data is shown in Figure 5.2.

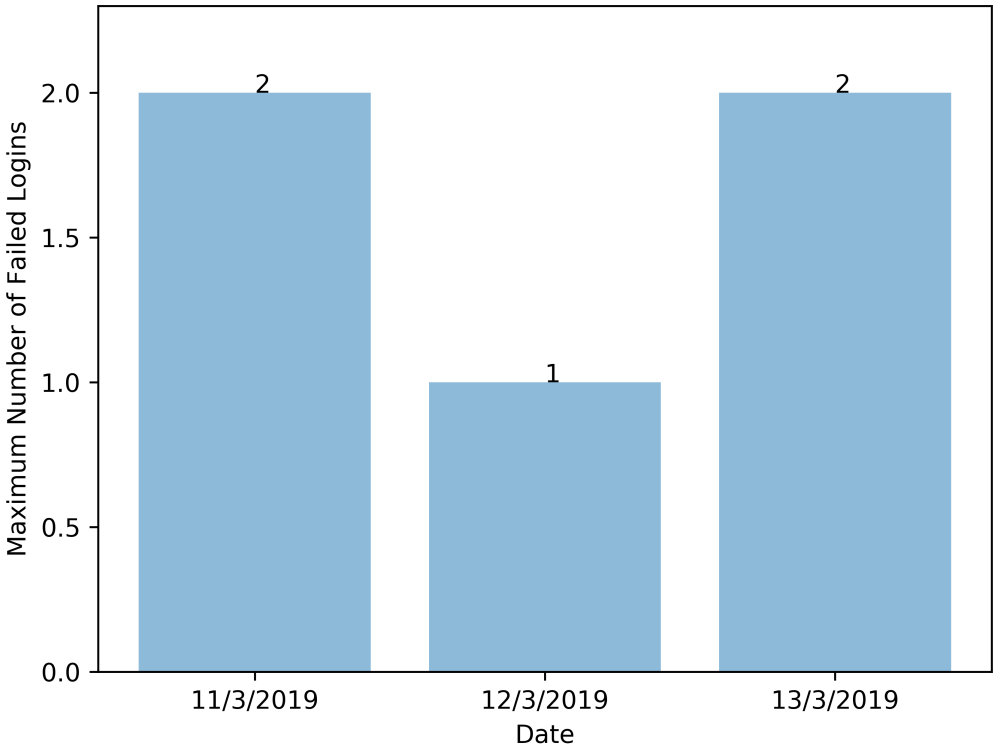

Figure 5.2: PasswordFails profile

We can see from the profile that the maximum number of normal failed logins that happened during the three days is two failed login attempts. This number can be used as a threshold in our model to detect attempted break-ins and masqueraders as explained in the model outlined in Algorithm 2. We have to mention that the normal behavior of failed logins can be more than 2, as we collected the data for only one user. Hence, setting the value of 2 as a threshold may result in a high number of false positives. To reduce the number of false positives in our tests, we will assign the value of 5 to the Threshold parameter. Deciding a value for the threshold when using the developed model in a real world environment would require collecting data for a large group of accounts during a long period of time to build the PasswordFails profile. The data collection scenario proposed here is only to establish a proof of concept in the current thesis. The proposed model has been implemented and run on the framework server. In addition, the model has been integrated with the framework by enabling the model to get access to the collected data stored in the framework

database.

5.1.5.3 Model testing

To test the model, we will use the same testbed, depicted in Figure 5.1, from the model building phase. We will conduct two types of tests. The first one is to run the model on the framework server and use the monitored computer for legitimate activities. While in the second test we will use a password cracking tool to crack the password of one of the accounts used to access the monitored host.

- **Regular login:**
  The purpose of this testing scenario is to check the behavior and the accuracy of the implemented model under normal work scenarios. We have designed three different test cases. The first test case involved providing wrong passwords for 2 times to test if the model would detect any failed logins when their count is under the threshold and also to test for false positives. The second test case involved providing wrong passwords more than 4 times. And the last test case involved providing wrong password 5 times and then providing the correct password to test if the model will detect masqueraders. The results are depicted in Table 5.1.

Table 5.1: Regular logins test cases

Test Case	**Results**
Test case 1	None
Test case 2	Failed logins warning
Test case 3	Intrusion alert

- **Compromised login:**
  In this testing scenario, we will test the ability of the proposed model to detect compromised login attempts when the monitored host undergoes login attacks. We will use **THC-Hydra** password cracking tool to crack the passwords for one of the accounts on the host monitored by the framework. Hydra is a password cracking tool that can perform rapid dictionary attacks against several protocols including SHH, Telnet, FTP and much more. Since the attacker machine is running Kali Linux, Hydra is already pre-installed. Hydra can be used to

perform a dictionary attack where you give it a list of passwords to try or it can perform a brute force attack trying all possible combinations of characters.

In this scenario, we will execute Hydra from the attacker machine to conduct password dictionary attack against one of the accounts of the hosts monitored by the framework. To perform dictionary attack using Hydra, we have created a passwords list of six random passwords that Hydra will try and attempt to access the targeted account. We decided to use a password dictionary of only six passwords rather than conducting brute force attack to test the ability of the implemented model to detect automated attacks when the number of login attempts exceeds the threshold even if the number of failed login attempts is not that large. The command we have used to perform our attack is shown if Figure 5.4.

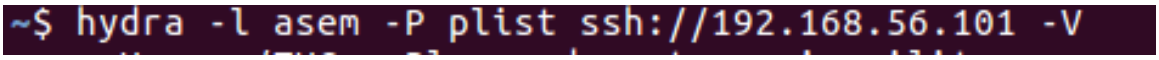

Figure 5.3: Hydra password cracking command

Linux records all the failed login attempts in the system logs and especially in the file "/var/log/wtmp". The framework reads the data from this file by executing the command *lastb* and provides them to the IDS model. A screenshot of the data extracted from Linux logs, that have been created when Hydra tried to access the monitored host using the dictionary attack, is shown in Figure 5.4.

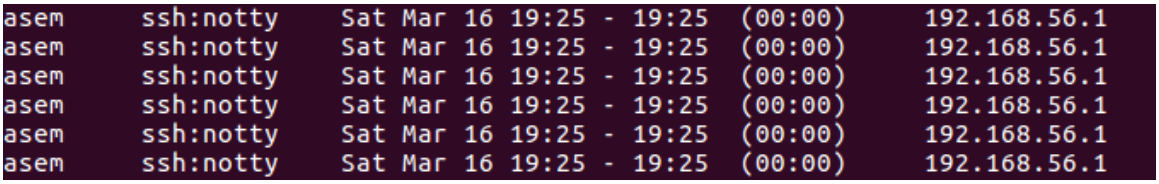

Figure 5.4: Failed logins logs

When we ran the attack command on the attacker node, the IDS model successfully detected the failed logins and generated "Failed login attempts warning." To test the ability of the IDS model on detecting intrusions, we have repeated the same scenario with a slight modification to the passwords list. We have added another password to the passwords list which is the correct password for the targeted account. The execution of the attack command resulted in an

"Intrusion alert" generated by the IDS model since the failed login attempts were followed by a successful login.

Chapter 6

Conclusions

6.1 Contributions Summary

This thesis proposed a security monitoring framework, for monitoring endpoint computing hosts—agentlessly—without the need to install any agent on the hosts being monitored. Because the existing security mechanisms for monitoring endpoints require installing agents that collect data from the monitored hosts, they suffer several drawbacks with regards to security, performance, scalability, maintenance and user acceptance, to name a few. The goal of the thesis was to find an agentless mechanism that provides the same functionality of the existing agent-based mechanisms while overcoming their drawbacks and challenges. The proposed framework architecture collects data items from the monitored endpoints remotely using standard and secure protocols and stores the collected data in a centralized database that is accessed by various security modules monitoring the security of the endpoints. Moreover, the proposed architecture enables the end users to extend the framework functionality easily, and can be used for monitoring large number of machines.

To validate the applicability of the proposed architecture, a prototype of the framework has been implemented at first, and then used to conduct evaluation experiments. The security of the framework has been evaluated and the results showed the resilience of the framework against potential attacks targeting its security properties. Moreover, experiments showed that increasing the number of the monitored hosts has very limited effect on the framework performance in collecting the data as well as the overhead on the framework server. Furthermore, as part of the evaluation process, we have built a real-time intrusion detection model that has been integrated

with the framework. The proposed model is for detecting abnormal behaviors on the monitored hosts based on the data collected by the framework from those hosts. The model was initially implemented to detect abnormal failed login attempts as well as masquerading logins that happen after launching password cracking attacks. Experimental results showed that the model was able to successfully detect all failed login attempts crossing a predefined threshold and masquerading attacks against user account on a monitored host.

6.2 Perspectives and Future Work

In general, this work showed that agentless architectures are promising and can be considered as a good solution to address the drawbacks of the current agent-based security monitoring architectures. With regard to the limitations of using agentless architecture to monitor endpoints, we can consider this point, to answer the third research question, based on our experiments in this thesis either those experiments related to implementation of the proposed framework or for the evaluation. In our implementation of the prototype, the only limitation that we can see is that not all platforms have ssh-server pre-installed, and ssh might not be supported by all mobile platforms. However, the implementation of an agentless architecture that supports several protocols, as mentioned in the main design of our proposed architecture, will address this limitation. The second point is whether there is any limitation in collecting the required data items. In our research, we collected a set of data items and we did not see any major limitation; however, there is a need to perform a large-scale collection of data to derive a concrete conclusion about limitations of agentless architecture in collecting all the required data to perform comprehensive security monitoring. Although the proposed framework addressed most of the functional, security, performance, scalability, and user acceptance challenges, there is still a number of challenges that need to be addressed in the future work. For example, the framework server might be considered as a single point of failure that can impact the whole monitoring system when it goes down for any reason. However, redundant and standby framework servers can be used to address this challenge in the current version of the framework architecture. Moreover, although the general concept behind agentless monitoring is to perform monitoring without installing agents on the monitored endpoints, it can be argued that the dependency on the standard protocols to collect data from the monitored endpoints requires specific daemons to be

running on the monitored endpoints, such ssh-server in our implemented prototype. Our claim is that the used protocols are standard protocols and are part of the normal functionality of the monitored endpoints, as they are used for several other purposes. In addition, the resources consumed by such daemons are trivial when compared to the aggregated resources used by agents.

There is a gap for more research that can be done with regards to advancing and enhancing agentless security mechanisms, and to answer several open questions that will help researchers in their quest to advance monitoring techniques, in general, and professionals to decide and choose the more suitable technique for their environments. The proposed framework can be used as a basis for conducting more research in this area. To build on top of this work, follow-up research will be done by another master's student to do more extensive validation work. In the current thesis, the focus has been on developing the framework and conducting relatively limited validation through a proof of concept. We have validated the framework by using it to collect a number of raw data that are related to the users activity, file system activity, processes spawned, network traffic, etc. However, in the future work, the framework will be used to collect large scope of data items that are needed for building more sophisticated security models. Moreover, the focus in this thesis was on using the framework agentlessly. A future work will focus on conducting a comprehensive data collection study that also considers the light agent mode in which the agentless architecture is used to collect most of the data required to perform anomaly detection while installing some light tools or agents on the monitored hosts to collect other kinds of data.

Much larger scale evaluation of the framework will be conducted, where the framework will be used in large networked environments with endpoints running various platforms (e.g., Linux, Unix, Windows, etc.). In the current work, we have theoretically evaluated the security of the proposed architecture based on specific threat model; however, the architecture should undergo an experimental security analysis as well, from the attackers' point of view. The future work will focus on exploring various attack vectors, and performing dynamic and static security analysis of the framework, in addition to launching several attacks targeting the framework security properties to evaluate its resilience against such attacks. The framework will be used for large scale monitoring, and based on this, datasets will be collected by involving a much larger user population and a diverse set of usage scenarios. Moreover, we plan to implement other anomaly detection models, and build more comprehensive and more sophisticated (e.g., using machine learning) intrusion detection models that are

integrated with the proposed framework.

One of the questions often raised by professionals and engineers is which monitoring method is ultimately the best, either agent-based or agentless monitoring? Based on our experience with designing and implementing the proposed framework, we can argue that answering such question requires conducting a large-scale evaluation of both agent-based and agentless techniques in various environments based on several factors that can involve security, reliability, performance, scalabilty, functionality, dependency, etc. Several articles addressed this question, such as [20]; however, all the proposed comparisons lack foundation rooted on studies and experiments. A future work can target this research question by using our proposed framework along with existing traditional agent-based monitoring frameworks or tools to perform large-scale studies in multiple environments under several circumstances. In addition to answering in what environments and under which circumstances would be combining both agentless and agent-based useful and achieves best results according to the factors of interests.

Bibliography

[1] Hyperic. https://www.vmware.com/products/vrealize-hyperic.html.

[2] MongoDB database. https://www.mongodb.com.

[3] Nagios open source. https://www.nagios.org.

[4] Public key authentication for ssh. https://www.ssh.com/ssh/public-key-authentication.

[5] Secure shell. https://en.wikipedia.org/wiki/Secure_Shell.

[6] Siem. https://en.wikipedia.org/wiki/Security_information_and_event_management.

[7] Software framework. https://en.wikipedia.org/wiki/Software_framework.

[8] Solarwinds. https://www.solarwinds.com.

[9] What is agentless monitoring? https://www.eginnovations.com/product/agentless-monitoring.

[10] Zabbix. https://www.zabbix.com.

[11] Konstantin Berlin, David Slater, and Joshua Saxe. Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 35–44. ACM, 2015.

[12] Morgan Brattstrom and Patricia Morreale. Scalable agentless cloud network monitoring. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 171–176. IEEE, 2017.

[13] Caitliningmar. Welcome to the ceilometer developer documentation! https://docs.openstack.org/ceilometer/latest.

[14] Jingsong Cui, Hao Xiang, Chi Guo, and Kun Hou. Agentless processes monitoring architecture on cloud platform. In *2014 International Conference on Cloud Computing and Big Data*, pages 1–7. IEEE, 2014.

[15] Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987.

[16] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[17] Asem Ghaleb, Issa Traore, and Karim Ganame. A framework architecture for agentless cloud endpoint security monitoring. In *2019 IEEE Conference on Communications and Network Security (CNS)*.

[18] Juan Gutierrez-Aguado, Jose M Alcaraz Calero, and Wladimiro Diaz Villanueva. Iaasmon: Monitoring architecture for public cloud computing data centers. *Journal of Grid Computing*, 14(2):283–297, 2016.

[19] Leslie Ingess. Why agentless security is the future of private cloud protection. `https://cloudtweaks.com/2013/02/why-agentless-security-private-protection`.

[20] Caitliningmar Koecher. Agent vs agentless: Why you should monitor (event) logs with an agent-based log monitoring solution. `https://www.eventsentry.com/blog/2017/03/agent-vs-agentless-why-you-should-monitor-event-logs-with-an-agent-based-log-monitoring-solution.html`.

[21] Asha Nagesh. Distributed network forensics using jade mobile agent framework. *Student Trade Show and Project Reports, Arizona State University*, pages 5–6, 2006.

[22] Caitlin O'higgins. The truth about agentless security - trend micro. `https://www.trendmicro.com/vmware/truth-agentless-security`.

[23] Wolfgang Pree. Meta patternsa means for capturing the essentials of reusable object-oriented design. In *European Conference on Object-Oriented Programming*, pages 150–162. Springer, 1994.

[24] Wei Ren. On a network forensics model for information security. In *Proceedings of the third international conference on information systems technology and its applications (ISTA 2004)*, pages 229–234, 2004.

[25] Kulesh Shanmugasundaram, Nasir Memon, Anubhav Savant, and Herve Bronnimann. Fornet: A distributed forensics network. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 1–16. Springer, 2003.

[26] Hongwei Tang, Shengzhong Feng, Xiaofang Zhao, and Yan Jin. Virtav: An agentless antivirus system based on in-memory signature scanning for virtual machine. In *2016 18th International Conference on Advanced Communication Technology (ICACT)*, pages 124–133. IEEE, 2016.

[27] Yongping Tang and Thomas E Daniels. A simple framework for distributed forensics. In *25th IEEE International Conference on Distributed Computing Systems Workshops*, pages 163–169. IEEE, 2005.

[28] Diangang Wang, Tao Li, Sunjun Liu, Jianhua Zhang, and Caiming Liu. Dynamical network forensics based on immune agent. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 3, pages 651–656. IEEE, 2007.