POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Digital Forensics in Corporate Simulations: A Study of Tool Efficacy and Analysis Techniques

Supervisor
prof. Atzeni Andrea

Candidate
Giacomo Zunino

April 2025

Summary

Digital forensics is a critical field within forensic science that focuses on the identification, acquisition, processing, analysis, and reporting of electronic data relevant to investigations. This discipline plays a critical role in responding to cyber attacks, allowing the identification, mitigation, and eradication of threats, and presenting key information to auditors, legal teams, and law enforcement following incidents. The forensic process involves several important steps: identification of potential evidence, preservation of electronically stored information (ESI), in-depth analysis of data objects, meticulous documentation of procedures, and effective presentation of findings to relevant stakeholders. This thesis explores the current application of digital forensics techniques within corporate environments. The aim is to develop practical strategies for their effective implementation and to identify areas that could benefit from improvement. The research highlights the alarming increase in cyber attacks, particularly against small and medium-sized businesses, which often lack adequate defenses. Exploring the latest advances in digital forensics, this study examines their integration into enterprise incident response strategies. To achieve these goals, controlled experiments were conducted in a virtualized environment, simulating real-world scenarios to evaluate the effectiveness of various forensic tools such as Autopsy. The scenarios addressed include a data breach involving an external attacker and an insider accomplice who helps carry out the attack, a phishing scheme through which the attacker gains access to employees' credentials to extract company data, and two distinct types of cyber attacks involving malicious software, both delivered via email and disguised as urgent updates. In the first one, the ransomware encrypts the contents of the victims' computers and leaves a ransom note with instructions, while in the second one, the malware remains on the computer and, periodically, sends the content of specific folders to the attacker. For each case examined, the configuration was set up in Docker to allow the simulation of attacks in a closed, isolated environment, providing flexibility in managing the various components involved. Following the setup and simulation phases, the components are isolated and subjected to a thorough analysis using forensic tools to investigate the origins of the attack and to reconstruct the sequence of events. The findings from this research aim not only to demonstrate the effectiveness of various digital forensics tools in corporate incident response but also to provide practical information for companies to improve their cybersecurity strategy.

Contents

Chapter 1

Introduction

Digital forensics is a branch of forensic science that focuses on identifying, acquiring, processing, analyzing, and reporting on data stored electronically that may be useful in an investigation.
A key aspect of this field is the analysis of suspected cyberattacks, with the objective of identifying, mitigating, and eradicating cyber threats. This makes digital investigation a critical part of the incident response process. Forensics is also useful in the aftermath of an attack, to provide information required by auditors, legal teams, or law enforcement.
It includes data from hard drives in computers, mobile phones, smart appliances, vehicle navigation systems, electronic door locks, and other digital devices.
To understand what computer forensics is, it is necessary to analyze its essential steps. The process begins with identification, where individual or devices likely to contain significant evidence are recognized. Following this, the preservation phase involves safeguarding relevant electronically stored information (ESI), capturing and documenting the crime scene, and ensuring all pertinent information is meticulously recorded.
Then, the evidence undergoes through analysis, which is a methodical examination that produces data objects, including system and user-generated files, and seeks specific answers and points of departure for conclusions. Forensic examiners meticulously examine the data, looking for files, system logs, and any fingerprints that might answer specific questions about the case.
Following the analysis, meticulous documentation is essential. This involves creating detailed reports that describe the entire forensic process (the forensic examiners must keep track of all the steps taken, including tools and methods used), from the identification and the preservation to the analysis. These reports must be clear, concise and replicable, allowing other qualified examiners to understand the investigation methodology and verify the results.
In the end, the presentation phase involves effectively communicating the outcomes to relevant stakeholders. This may include presenting evidence in court, delivering a report to the internal management, or briefing investigators from other agencies. The presentation should be tailored to the specific audience, ensuring clarity and comprehension while respecting legal and ethical guidelines.
The importance of digital forensics in business environments is underlined by alarming statistics on cyber attacks. According to *stationx*, 48% of small and mid-sized businesses (SMBs) worldwide have faced a cybersecurity incident in the past year, with 25% reporting multiple incidents during the same period. Another interesting fact is that from 2020 to 2022, the number of cyber attacks targeting small and mid-sized businesses increased by 150%, escalating to 31,000 attacks per day worldwide. One important factor to consider following a cyber attack is its impact on the company. According to a research by *stationx*, the global average cost of a data breach now exceeds $4 million USD. While this figure is significantly lower for smaller businesses, a major breach can still have a much more severe financial and operational impact, including significant losses in customer trust.
According to *Embroker*, cybercrime is projected to cost companies worldwide approximately $10.5 trillion annually by 2025, up from $3 trillion in 2015, reflecting a year-over-year growth rate of 15%. Cyber attacks are becoming more frequent, targeted, and complex across all types of businesses, with a particular emphasis on small to medium-sized enterprises. This trend is partly due to the fact that 43% of cyber attacks are aimed at small businesses, while only 14% of these

businesses are sufficiently prepared to defend themselves. A cyber attack not only disrupts normal operations but can also cause severe damage to critical IT assets and infrastructure, often leaving small businesses unable to recover without sufficient budget or resources. Small businesses are particularly vulnerable, as evidenced by the Ponemon Institute's State of Cybersecurity Report, which highlights that many small to medium-sized businesses worldwide have recently experienced cyber attacks. The report reveals that 45% of these businesses find their security measures ineffective in mitigating attacks, 66% have encountered a cyber attack in the past 12 months, and 69% believe that cyber attacks are becoming more targeted. The most common types of attacks on small businesses involve phishing and social engineering, compromised or stolen devices, and credential theft.

This thesis aim to analyze the current State of the Art in digital forensics techniques commonly used within corporate environments. By exploring these techniques, the research aims to derive practical strategies for their effective application and identify potential areas for improvement. Furthermore, the latest advances in digital forensics research will be investigated to explore their potential integration into corporate incident response strategies.

To achieve these objectives, digital forensics experiments will be conducted within a controlled, virtualized environment. This environment will allow the simulation of real-world scenarios and test the efficacy of various forensics tools (like Autopsy, Wireshark, Volatility, etc...).

This research, by analyzing the effectiveness of current and emerging digital forensics techniques, has the goal to provide companies with practical recommendations for optimizing their incident response capabilities.

Chapter 2

Background

In this chapter there is a description of the tools used to perform the research, such as virtualization and digital forensic analysis tools.

2.1 Virtualization

Virtualization is a process that allows for more efficient use of physical computer hardware and is the foundation of cloud computing. It uses software to create an abstraction layer over computer hardware, enabling the division of a single computer's hardware components, such as processors, memory and storage, into multiple virtual machines (VMs). Each VM can run its own operating system (OS) and behaves like an independent computer, even though it is running on just a portion of the actual underlying computer hardware.
Virtualization offers significant advantages to data center operators and service providers by enhancing resource efficiency, simplifying management, minimizing downtime, and speeding up provisioning. In terms of resource efficiency, virtualization overcomes the limitations of traditional IT practices where each application server required a dedicated physical CPU. Server virtualization allows multiple applications to run on a single physical machine, each within its own virtual machine with a dedicated operating system, without compromising reliability. This maximizes the utilization of the physical hardware's computing capacity. Virtualization also simplifies management by replacing physical machines with software-defined VMs, making it easier to manage IT policies through software, and simplifying the automation of IT service management workflows. Virtualization also enables the implementation of security policies that enforce specific configurations based on the role of each VM. Operating system and application crashes can cause downtime and impact user productivity. Admins can run multiple redundant virtual machines side by side and failover between them when problems occur. Virtual machines can be snapshot at various points in time. Snapshots capture the VM's state, including data and configurations and in case of a security breach or system failure, administrators can quickly restore the VM to a previous snapshot, minimizing downtime. Finally, virtualization accelerates the provisioning process, because buying, installing and configuring hardware for each application is time-consuming. If the hardware is already in place, provisioning virtual machines to run all the applications is significantly faster.
In virtualization there are two important concepts, and they are virtual machines and hypervisors. A **virtual machine** is a software-defined computer that runs on a physical computer with a separate operating system and computing resources. The physical computer is called the host machine and virtual machines are guest machines. Multiple virtual machines can run on a single physical machine. Virtual machines are abstracted from the computer hardware by a hypervisor. The **hypervisor** is a software component that manages multiple virtual machines in a computer. It serves as an interface between the VM and the underlying physical hardware, ensuring that each VM has access to the necessary physical resources for execution. Additionally, it prevents the VMs from interfering with one another by safeguarding their memory space and compute cycles from overlapping. There are two types of hypervisors [1]. The *type 1 or "bare-metal" hypervisor*,

is a hypervisor program installed directly on the computer's hardware instead of the operating system and it interact with the underlying physical resources, replacing the traditional operating system altogether. Type 1 hypervisors have better performance and they most commonly appear in virtual server scenarios. *Type 2 hypervisors* run as an application on an existing OS and they are suitable for end-user computing. Type 2 hypervisors carry a performance overhead because they must use the host OS to access and coordinate the underlying hardware resources.
Among the most common hypervisors, there are Hyper-V, KVM, and VMware [2]. **KVM** is an open source Linux based type 1 hypervisor that can be added to most Linux operating systems and it supports most common Linux operating systems, Solaris, and Windows. **Hyper-V** provides several advanced features, including live migration, storage migration, and VM replication (Replica), which enhance system flexibility and resilience. It also supports dynamic memory allocation and an extensible virtual switch, contributing to efficient resource management and network configuration. **VMware** led the market in developing innovative features such as memory over commitment, vMotion, Storage vMotion, Fault Tolerance, distributed resource scheduler (DRS) for VM "load balancing" and more.
There are different types of approaches to virtualization that differ primarily in how they manage system resources and interact with the host and guest operating systems [3]. These diverse virtualization methods each offer distinct advantages in terms of efficiency, performance, and resource management, making them suitable for different use cases and environments. **Full Virtualization**, for example, involves a *Virtual Machine Monitor* (VMM) that runs on a host operating system, typically within user space. In this setup, both applications and the guest operating system operate on virtualized hardware provided by the VMM, which emulates the underlying physical devices. This approach is user-friendly, as it allows the installation and use of guest operating systems in a manner similar to running them directly on hardware, though performance can be reduced by up to 30%. **OS-layer virtualization**, on the other hand, runs multiple instances of the same operating system concurrently, with the host OS itself being virtualized. The virtualized environment shares a common operating system image, simplifying system administration through dynamic allocation of resources such as memory, CPU, and disk space. While this method is generally more efficient and offers good isolation, it does not allow for the use of different guest OS types, as they must match the host OS. In contrast, **hardware-layer virtualization** operates directly on the hardware, providing strong isolation and high performance by managing access to physical resources through the VMM. **Para virtualization** requires modifications to the guest operating system to operate within the virtualized environment, and while the guest OS is aware of the virtualization, this approach can achieve near-native performance due to its simplified VMM. In terms of device interaction, para virtualization functions similarly to full virtualization, relying on physical device drivers for virtual devices. **Application virtualization** allows users to run applications in isolated virtual environments without fully installing them on the system, providing a lightweight and efficient solution for application deployment. Furthermore, **resource virtualization** includes a variety of methods including the aggregation of multiple resources into larger pools or the partitioning of single resources, such as storage, into smaller, more manageable units. **Storage virtualization**, specifically, integrates the functions of physical storage devices, such as network-attached storage (NAS) and storage area networks (SAN), enabling the consolidation of multiple storage hardware within a data center, regardless of different vendors or types [4]. This type of virtualization, in the end, offers a simplified and unified storage system for users. **Network virtualization** uses software to create a *view* of the network that an administrator can use to manage the network from a single console. It abstracts hardware elements and functions (for example connections, switches, routers and firewalls) and abstracts them into software running on a hypervisor. The network administrator can modify and control these elements without touching the underlying physical components, which dramatically simplifies network management [1].
From a security perspective, virtualization plays an important role, as it ensures the isolation of each VM from other machines and the physical environment, limiting the impact of compromised applications. For example, in case there are VMs infected with malware, they can be rolled back to a point in time (called a snapshot) when the VM was uninfected and stable; they can also be more easily deleted and re-created. Virtualization also provide isolation, which means that any malicious activity or compromise within one VM is unable to spread to others, enhancing overall system security. So, even if a malware infiltrates one VM, it remains contained and cannot easily

affect neighboring VMs. Another positive aspect is that a controlled and isolated environment allows for the safe execution of untrusted or potentially malicious applications. This is called *sandboxing* and it serves as a critical security measure, creating a controlled and isolated space in which untrusted or potentially malicious applications can be safely executed [5]. Virtualization also presents some security challenges and it requires the implementation of strong cybersecurity measures, including firewalls and intrusion detection systems, alongside regular security audits. Although virtual machines can be easily replicated and transferred to another server in the event of a data breach or virus attack, the compromise of one virtual machine can potentially impact other virtual machines on the same server and such attacks can be difficult to detect [1]. If an attacker compromises a hypervisor, they potentially own all the VMs and guest operating systems. Because hypervisors can also allow VMs to communicate between themselves without touching the physical network, it can be difficult to see their traffic, and therefore to detect suspicious activity.

In the field of digital forensics, virtualization can be used by investigators to conduct digital forensic analysis, because it allows the creation of secure and isolated environments to examine, for example, an image of a suspect's hard drive and it also allows the investigators to create multiple virtual copies of the evidence so they can run different digital forensics tools simultaneously without affecting the original data. Virtual machines are useful tools in forensics since they can track and record activity trails of users and produce seamless recreation of crime scenes for further forensic examinations. Law enforcement officers collect images from a suspect's native environment and analyze these files in virtual machines to see how perpetrators, along with their evidence, respond and react in their natural states [6].

Virtual machines can be a valuable tool in forensic investigations and can also be used by attackers to thwart forensics investigations just as easily in order to cover their own tracks.

2.2 Autopsy

Autopsy is a digital forensics platform used for conducting in-depth examinations of digital devices and file systems. It is a tool that assists in the investigation of what happened on a computer by analyzing data and it supports various file formats, like disk image (img, dd, raw, 001, e01, etc) or VM file (vmdk and vhd), local drive and logical files.
Autopsy provides ingest modules that are used to analyze the data in a data source. These modules conduct comprehensive analysis of the files and parse their contents [7].
Recent Activity extracts information about recently accessed files, applications, and system events from the registry and other system artifacts. It can help investigators identify recently used files or programs, which can be crucial in determining the timeline of events. It also runs Regripper (written in Perl, is the fastest, easiest, and best tool for extracting/parsing information, like key, values and data, from the registry and presenting it for analysis) on the Registry hive. This allows the investigator to see what activity has occurred in the last seven days of usage, what web sites were visited, what the machine did, and what it connected to. There is nothing to configure and there are no run-time settings for this module. The results show up in the tree under "Extracted Content".
The **Hash Lookup** module calculates MD5 hash values for files and then it compares them with known databases (added during the configuration) to determine if the file is notable (malicious), known (in general), or unknown. The results show up in the tree as "Hashset Hits", grouped by the name of the hash set. If the hash set hits had associated comments, it will be possible to see them in the "Comment" column in the result viewer along with the file hash.
The **File Type Identification** module identifies files based on their internal signatures and does not rely on file extensions. Autopsy uses the Tika library (a content analysis toolkit that detects and extracts metadata and text from over a thousand different file types, such as PPT, XLS, and PDF) to do its primary file ID detection and that can be customized with user-defined rules. This modules should be enabled because other many modules depend on its results to determine if they should analyze a file (like Extension Mismatch Detector Module). No configuration is necessary with this module unless custom types need to be defined. The results can be seen in the views area of the tree, under Views → File Types → By MIME Type.
The **Extensions Mismatch Detection** module uses the results from the File Type Identification and flags files, ignoring 'known' (NSRL) ones, that have an extension not traditionally associated with the file's detected type. The MIME types and file extensions for each MIME type can be customized in "Tools" under "Options" and then "File Extension Mismatch". In the ingest settings, users can select to run on all files, all files except text files, or only multimedia or executable files. Additionally, users can choose to skip all files without an extension and to skip any known files identified by the hash lookup module, if it is enabled. The results are shown in the Results tree under "Extension Mismatch Detected".
The **Embedded File Extractor** module opens ZIP, RAR, other archive formats, Doc, Docx, PPT, PPTX, XLS, and XLSX and sends the derived files from those files back through the ingest pipeline for analysis. This module expands archive files to enable Autopsy to analyze all files on the system and it enables keyword search and hash lookup to analyze files inside of archives. Each file extracted shows up in the data source tree view as a child of the archive containing it and as an archive under "Views", "File Types", "Archives". In case the module encounter an encrypted archive, it will generate a warning bubble in the bottom right of the main screen and, after the ingest, it will be possible to attempt to decrypt these archives (if the password is known). After entering the password, it is possible to select which ingest modules to run on the newly extracted files.
The **Picture Analyzer** module extracts EXIF (Exchangeable Image File Format) information from ingested pictures, which can include geolocation data, time, date, camera model, settings (such as exposure values and resolution), and other details. The extracted attributes are added to the Blackboard, providing information on where and when a picture was taken and clues about the camera used. The module also converts HEIC/HEIF images to JPG while preserving their EXIF information, which is processed and saved as it would be for standard JPG images. The results are displayed in the Results tree.
The **Keyword Search** module searches for specific keywords or phrases within file content, metadata, and other attributes and it also supports manual text searching after the ingest has

completed. Autopsy tries to extract the maximum amount of text from the files being indexed. Initially, it attempts to extract text from supported file formats, including plain text files, MS Office documents, PDF files, emails, and various others. If the file format is not supported by the standard text extractor, Autopsy resorts to a string extraction algorithm. The Ingest Settings for the Keyword Search module allow the user to enable or disable the specific built-in search expressions, Phone Numbers, IP Addresses, Email Addresses, and URLs. The user, from "Global Settings", is also able to add custom keyword groups. With the release of Autopsy 4.21.0, two types of keyword searches are supported: Solr search with full-text indexing and the built-in Autopsy "In-Line" Keyword Search. Solr search provides users with the flexibility to perform ad-hoc manual text searches after the ingest process is complete, but it can significantly slow down the ingest speed for large data sources and cases. In contrast, the "In-Line" Keyword Search requires all search terms to be specified before the ingest begins, without the option for ad-hoc searches on the entire extracted text after the ingest process is complete. The module will save the search results regardless whether the search is performed by the ingest process, or manually by the user. The saved results are available in the Directory Tree in the left hand side panel (Analysis Results → Keyword Hits).

The **Email Parser** module identifies files in MBOX, EML, and PST formats based on their file signatures, extracts the emails, and adds the results to the Blackboard. It is also able to retrieve the attachments and add them as derived files. The results show up in the "Results", "E-Mail Messages" portion of the Tree Viewer.

The **Encryption Detection** module searches for files that could be encrypted using both a general entropy calculation and more specialized tests for certain file types. The settings can be configured at runtime and they only effect the tests that are based on entropy. A minimum entropy can be set, depending on how many false hits are being produced. It is also possible to run the test only on files whose size is a multiple of 512, which is useful for finding certain encryption algorithms. Files that pass the tests are displayed in the Results tree under "Encryption Detected" or "Encryption Suspected." Typically, if the test involved identifying a specific header or file structure, the result will be listed as "Encryption Detected," with the type of encryption noted in the Comment field. If the test was based on file entropy, the result will be shown as "Encryption Suspected," with the calculated entropy value provided in the Comment field.

The **Interesting Files Identifier** module enables the automatic flagging of files and directories that match a predefined set of rules, and it can be useful for identifying files with specific names or paths within a data source, or for targeting files of a particular type that are of interest. Files that match any of the rules in the enabled rule sets will be shown in the Results section of the Tree Viewer under "Interesting Items" and then the name of the rule set that matched.

The **Central Repository** module allows a user to find matching artifacts both across cases and across data sources in the same case. It is a combination of an ingest module that extracts, stores, and compares properties against lists of notable properties, a database that stores these properties, and an additional panel in Autopsy to display other instances of each property. The central repository database can either be SQLite or PostgreSQL. The results can be seen in two places: the Content Viewer, which shows all matching properties from other cases or data sources for each file or artifact, and the Analysis Results section, which includes files or results that match properties previously marked as notable, have been seen before, or are unique.

The **PhotoRec Carver** module carves files from unallocated space in the data source and sends the files found through the ingest processing chain. This can help a reviewer discover more information about files that used to be on the device and were subsequently deleted. The results of carving show up on the tree under the appropriate data source with the heading "$CarvedFiles".

The **Virtual Machine Extractor** module adds any virtual machines it finds in a data source to the case as new data sources; this includes virtual machine disk (.vmdk) files and virtual hard drive (.vhd) files.

The **Data Source Integrity** module has two purposes. If the data source has associated hashes, whether user-entered or embedded in an E01 file, the module verifies these hashes. If no hashes are associated with the data source, the module calculates them and stores the results in the database.

The **Android Analyzer (aLEAPP)** and the **iOS Analyzer (iLEAPP)** modules run, respectively, aLEAPP and iLEAPP (android and iOS Logs Events And Protobuf Parser), and converts the results into results that can be viewed in Autopsy. The module will run on .tar/.zip files

found in a logical files data source or a disk image and the result will appear in the Tree Viewer under Results → Extracted Content.
The **YARA Analyzer** module uses a set of rules to search files for textual or binary patterns. It was designed for malware analysis but can be used to search for any type of file. The results are displayed in the Results tree under "Extracted Content".
The **GPX Analyzer** module enables the import of GPS data from GPX files. GPX (the GPS Exchange Format) is a light-weight XML data format for the interchange of GPS data (waypoints, routes, and tracks) between applications and Web services on the Internet. The results are show in the Results tree under "Extracted Content".
Plaso is a framework for running modules to extract timestamps for various types of files. The Plaso ingest module runs Plaso to generate events that are displayed in the Autopsy **Timeline**. The Plaso ingest module runs dozens of individual parsers and can take a long time to run. In testing, the slowest parsers by far were winreg, pe, and chrome_cache. chrome_cache is always disabled as it duplicates events created by the Recent Activity Module. The Plaso events will be shown in the Timeline.
Autopsy organizes data by case and each case can have one or more data sources. To create a case, it is possible to use either the "Create New Case" option on the Welcome screen or from the "Case" menu. In this step, the system requires the input of the case name and a designated directory for storing the case results. This process is followed by a step in which an input data source must be added. The user must select the type of input data source, such as an image (E01 and raw (dd) files), local disk (select one of the detected disks and then Autopsy will add the current view of the disk to the case, like a snapshot of the meta-data), or logical files and folders. Subsequently, the location of the chosen data source must be provided. In the next phase there is the list of ingest module to configure and then run on the data that must be analyzed. Once the ingest modules begin analyzing the data source, the main analysis interface becomes accessible. At this stage, it is possible to search for specific items, navigate to particular folders, or see the results generated by the ingest modules.
There are several specialized interfaces available to enhance the analysis of the data source contents. The **Timeline** feature, accessible via the "Tools" menu, toolbar, or the "Timeline" button, organizes file system and other events chronologically, employing various display techniques for effective analysis. The **Image Gallery** interface is designed to present pictures and videos from the data source, organized by folder. Files are displayed as soon as they are hashed and EXIF data is extracted. This interface can be accessed from the "Tools" menu or by selecting the "Image Gallery" button. The **Communications** interface highlights the most frequently contacted accounts and the messages exchanged, allowing for focused analysis of specific relationships or communications within a certain date range. This interface is also accessible through the "Tools" menu or the "Communications" button. The **Geolocation** panel provides a map with markers indicating all geolocation results found in the case, accessible via the "Tools" menu or the "Geolocation" button. The **Discovery** panel facilitates the search for different types of data within a case, presenting it in a format that is easy to review. It can be accessed through the "Tools" menu or the "Discovery" button. Lastly, the **Personas** panel is used for creating and managing personas, allowing the association of one or more accounts with specific names and other relevant data. This feature can be accessed via the "Tools" menu or the "Personas" button.
Among the positive aspects of Autopsy, it is possible to highlight its user-friendly interface, its ability to carry out comprehensive analyzes and the ability to manage large-scale data investigations. Additional benefits are the fact that it is an open-source tool, which makes it accessible and customizable, and it provides in-depth and detailed reports useful for legal and investigative purposes.

2.3 Volatility

Volatility is an open-source memory forensics framework used for analyzing volatile memory (RAM) from computer systems. Digital forensic investigators use this tool to extract and analyze information such as running processes, open network connections, registry keys, and other valuable data from a system's memory [8].
Volatility splits memory analysis down to several components and the main ones are memory layers, templates and objects, and symbol tables [9]. A **memory layer** is a set of data that can be accessed by requesting data at a specific address. At its lowest level, this data is stored on a physical medium, such as RAM. In the early days of computing, systems addressed memory locations directly. However, as memory capacity grew and became more challenging to manage, most architectures transitioned to a "paged" memory model. In this model, the available memory is divided into fixed-sized pages, and to simplify memory management, programs can request any address, and the processor will use a mapping system to translate the virtual address into its corresponding physical address within the system's actual memory. Volatility can work with these layers as long as it knows the map. The automagic (certain setup tasks) that runs at the start of every volatility session often locates the kernel's memory map, and creates a kernel virtual layer, which allows for kernel addresses to be looked up and the correct data returned. There can be several maps, and in general there is a different map for each process (although a portion of the operating system's memory is usually mapped to the same location across all processes). These maps may use the same virtual address but point to different parts of physical memory. This also allows for the possibility that two processes could share memory by having a virtual address mapped to the same physical address as another process. Once contiguous chunks of memory can be addressed and a virtual address can be translated into the actual data used by the processor, it becomes possible to extract **objects** by applying a **template** to the memory layer at a specific offset. A template contains all the information about the structure of an object without being populated by actual data. It can define the size of the structure and its members, indicate the position of a particular member within the structure, and suggest what various values in that field might represent, though it does not specify the actual content of any particular member. Most compiled programs know their templates and define the structure (and location within the program) of these templates as a Symbol. A symbol typically consists of an address and a template, and it can be used to reference either independently. Lookup tables containing these symbols are often generated as debugging information during the program's compilation. Volatility 3 provides access to these through a **SymbolTable**, many of which can be collected within a Context as a SymbolSpace. A Context can store only one SymbolSpace at a time, although a SymbolSpace can store as many SymbolTable items as necessary.
Analysts can use Volatility for memory forensics by leveraging its unique plug-ins to identify rogue processes, analyze process dynamic link libraries (DLL) and handles, review network artifacts, and look for evidence of code injection. Volatility has multiple plug-ins that enable the analyst to analyze RAM in 32-bit and 64-bit systems. These plug-ins also allow the Digital forensics and incident response (DFIR) analysts to extract the process, drives, and objects, and check for the rootkit signs running on the device of interest at the time of infection.
Once the random-access memory (RAM) artifacts found in the memory image are acquired, the next step is to analyze the obtained memory dump file for forensic artifacts. The memory image analysis can determine information about the process running, created files, users' activities, and the overall state of the device of interest at the time of the incident.
One of the powerful features of Volatility is its ability to detect hidden or deleted processes. Unlike basic operating system tools and other security applications that usually provide access to runtime data, which can be easily manipulated by malicious software, Volatility is a tool capable of directly analyzing data in RAM, enabling it to detect such manipulations. Therefore, Volatility's most critical feature is its plugins that analyze process activity. The plugins and their functions are: pslist, pstree and psscan [8]. When using these plugins in Volatility, certain key headers are essential to understand. These include the Process ID number (PID), the Parent Process ID number (PPID), the name of the running process (ImageFileName), the hexadecimal value indicating the memory location where the process is running (Offset), the number of threads for a process (Threads), the time the process started (CreateTime), and the time it ended (ExitTime). Reviewing the process names can help identify anything unusual. If a process name stands out,

a quick online search can determine whether it looks legitimate or requires further attention.
The **pslist** command tracks the operating system's process list and catalogs all processes known by the operating system. The output from using pslist can be large, so it may be worth using some specific syntax, like *python3 vol.py -f <filename> windows.pslist | less* and *python3 vol.py -f <filename> windows.pslist > output.txt*, to tab through the output or dump it into a text file.
The **pstree** command takes the output from pslist and organizes it in a tree-like format to clearly display parent-child relationships among processes. This facilitates the identification of suspicious process activities by allowing visibility into which processes initiated others, such as 'cmd.exe' or 'powershell.exe', making it possible to assess whether the activity appears legitimate. Another advantage of using 'pstree' is its ability to reveal how attackers often disguise their malware by naming its running process after legitimate Windows processes, such as 'svchost.exe'. This tactic allows the malware to blend in with other system processes, making it more difficult to detect.
The **psscan** command directly scans physical or virtual process blocks in memory and it has the ability to find terminated or disconnected (hidden) processes as well. This allows for a more comprehensive process list to be obtained. After executing the pslist and psscan commands, the two outputs should be compared side by side to identify any differences between the processes. A more detailed analysis is then performed on the processes that do not match to determine whether they are malicious or not.
When a RAM dump is captured any network connections at the time the capture was taken will also be stored within the captured memory. This is is great for incident responders as any malicious network connections can be identified such as the source port, destination IP, destination port, and the process that the network activity relates to.
The **netscan** command [10] is the one used to view the network connections associated with the RAM dump that is being analyzed. The output of netscan includes 10 columns that provide detailed information about network connections. These columns indicate the memory location (Offset), the network protocol used (Proto), and the source and destination addresses (LocalAddr and ForeignAddr) and ports (LocalPort and ForeignPort) of the connection. Additionally, they reveal the state of the network connection (State), such as whether it is established, closed, or listening, along with the Process ID (PID) and the account associated with the process (Owner). The time when the network connection was initiated (Created) is also recorded.
Volatility can also be used to detect malware, which is often packed so that the code written by the malware author is obfuscated in order to make it harder to be identified. When a piece of malware is packed it will need to unpack itself, it does this in memory which is great for Volatility as that is exactly what the tool is designed to analyze. It is important to know that the packed malware typically unpacks itself by creating a child process and injecting the unpacked executable or malware into this new process.
To search for injected code with Volatility, the **malfind** feature is used [10]. This feature provides a list of processes that Volatility suspects might contain injected code, based on header information displayed in hexadecimal, permissions, and extracted assembly code. However, just because a process appears in the output does not guarantee that it is malicious. The key points to understand are the PID, the process name, the protection, and hexdump disassembly. The Protection relates to the output 'PAGE_EXECUTE_READWRITE' and this means that the process has execute, read and write permissions. This is common for malware as it needs to be able to execute itself or other executable it downloads/drops, it needs to read files on the device it has compromised and it needs to be able to write files to disk. So any processes captured that have these permissions will be displayed in the malfind output. The hexdump disassembly part shows only the first X bytes of the memory page (represented in hexadecimal and next to that the same values in ASCII) and in some cases it is not enough to find some code, so a solution could be use the dump option (or external tools) and look for hashes on the internet.
With Volatility, it is possible to perform a memory forensic analysis on a memory dump derived from an infected system. The initial step involves retrieving basic information about the system with the command *windows.info*, followed by the command *pslist* to obtain a list of processes. Suspicious processes can be identified from this list based on their names and number of threads. The *pstree* command provides a clearer view of the relationships between processes, making it easier to detect suspicious activities. Another important step is to check the network connections at the time the capture was taken and this can be done with the *netscan* command. If a specific process is suspected, a filter can be applied to search all connections opened by that process.

Otherwise, the full list of connections is available. When a connection initiated by a suspicious application is identified, it is a good idea to verify the associated IP address on websites like VirusTotal to determine if it is linked to known malware.

2.4 Wireshark

Wireshark is an open-source and powerful network traffic analysis tool that is useful in digital forensics, as real-time forensics can be performed simply by setting up Wireshark on a portable disk, in order to assist with incident response and triage (which is the process of sorting and categorizing digital evidence based on its relevance and importance to an investigation).
This forensic tool allows investigators to comprehend the situation quickly and stop the attack while gathering evidence and data to stop future attacks.
Wireshark listens to a network connection in real time and then grabs entire streams of traffic and allows to filter the collected traffic by applying filter in order to obtain just the necessary information. Wireshark is a packet sniffer and analysis tool and it captures network traffic from Ethernet, Bluetooth, wireless (IEEE.802.11), token ring, and frame relay connections, among others, and stores that data for offline analysis. Wireshark provides the ability to apply filters either before capturing network data or during the analysis phase, enabling a focused examination of specific elements within a network trace. For example, filters can be configured to display only TCP traffic between two IP addresses or to show packets sent from a particular computer.
The filters in Wireshark are one of the primary reasons it has become the standard tool for packet analysis. Wireshark can be used to understand how communication takes place across a network and to analyze what went wrong when an issue in communication arises. It assists network administrators in troubleshooting network problems, helps security engineers analyze security issues, supports QA engineers in verifying applications, helps developers debug protocol implementations, and enables network users to gain insights into specific protocols. By default, Wireshark only captures packets going to and from the computer where it runs. By checking the box to run Wireshark in promiscuous mode in the capture settings, it is possible to capture most of the traffic on the LAN [11].
To correctly use Wireshark, it is important to understand the protocols processed at each OSI layer [12]. The OSI model is structured in layers, each responsible for different protocols. At the Application Layer (Layer 7), protocols such as SMTP, HTTP, FTP, POP3, and SNMP handle end-user network processes. The Presentation Layer (Layer 6) deals with data representation and encryption through protocols like MPEG, ASCII, SSL, and TLS. In the Session Layer (Layer 5), protocols like NetBIOS and SAP manage sessions between devices and ensures connections between end-points are continuous and uninterrupted. The Transport Layer (Layer 4) ensures reliable data transfer through TCP and UDP. The Network Layer (Layer 3) handles routing and addressing with protocols including IPv4, IPv6, ICMP, IPsec, ARP, and MPLS. The Data Link Layer (Layer 2) facilitates data transfer between adjacent network nodes using protocols like PPP, Frame Relay, and ATM. Lastly, the Physical Layer (Layer 1) involves the transmission of raw data over physical mediums, with protocols such as RS232 and 100BaseTX. Understanding the function of each layer enables a more focused approach to network troubleshooting. For example, if there is an issue with internet connectivity, it might indicate a problem at the Network Layer (Layer 3), where router-related data is processed. Wireshark can then be employed to further investigate this hypothesis, identifying the specific layer and protocols where errors are occurring.
When opening Wireshark, the interface presents a screen displaying all available network connections that can be monitored. Additionally, there is a capture filter field that allows the user to specify and filter the network traffic of interest. By using shift+left-click, multiple network interfaces can be selected simultaneously. Once the desired network interface has been chosen, the capture process can be initiated.
Wireshark provides a detailed view of packet data through three distinct panes. The top pane, known as the Packet List, displays all captured packets in sequence. When a packet is selected, the other two panes change to show you the details about the selected packet. This selection also allows for identifying whether the packet is part of an ongoing conversation. In the Packet List, each packet is numbered in the order of capture, with brackets indicating its involvement in a conversation. The time column shows the elapsed time since the start of the capture when the packet was recorded, with the option to adjust this display through the Settings menu. The source and destination columns indicate the sending and receiving addresses of the packet, respectively, while the protocol column identifies the packet type, such as TCP, DNS, DHCPv6, or ARP. The length column provides the packet size in bytes, and the info column offers additional details about the packet's contents, varying by packet type. The middle pane, Packet Details, shows as much

readable information as possible based on the packet type. Users can right-click within this pane to create filters based on the highlighted content. The bottom pane, Packet Bytes, displays the packet in its original hexadecimal form. When looking at a packet that is part of a conversation, it is possible to right-click the packet and select "Follow" to see only the packets that are part of that conversation.
In case of an analysis of malicious HTTPs traffic, the investigation will start by looking for successful TLS handshakes and this can be done by using "tls.handshake.type eq 1". Since the traffic is encrypted, decryption will be performed using the appropriate decryption key. To decrypt encrypted traffic, an encryption key log file is required [13]. This file is a text document containing unique key pairs necessary for decrypting the encrypted session. As for generating the key log file, there are different ways to do it depending on the operating system that is used. The first step is to close all Firefox and Chrome browsers. In the case of Windows, the next thing to do is open *environment variables* in order to edit the environment variables for your account. In the Environmental Variables pop-up window, under User variables for <user name>, by clicking on *new*, it is possible to add a new variable. The first thing to enter is the *variable name*, which in this case is SSLKEYLOGFILE, and then the *variable value*, that contains the full path name for the TLS log file (e.g. C:\Users\user1\Desktop\tlskeylog.txt). For macOS or Linux, the OS terminal need to be opened, in order to have access to command line. At this point, the environment variable must be set with the following command: export SSLKEYLOGFILE="\Users\<account_name>\sslkeyfile" (macOS) or export SSLKEYLOGFILE="\Home\<account_name>\sslkeyfile" (Linux). After the new environment variable has been created, it is possible to start Wireshark and open browsers like Chrome or Firefox. When accessing a web application, the TLS session keys are logged in the file with the path specified in the environment variable created before.
This key log file will contain key pairs that are automatically generated for each session when a connection is established with a SSL/TLS-enabled webpage, allowing the browser to write the keys to this file as the user navigates the web. Since SSL/TLS key pairs are created at the time of connection, it is important to capture the keys during the traffic capture process. Failing to do so will prevent the creation of a suitable key log file needed for decrypting the captured traffic. The key log files can be managed using the "right-click" menu or by navigating to "Edit → Preferences → Protocols → TLS" in the settings. The packet details and bytes pane presents the data in various formats for the investigation.
Once the traffic has been decrypted, it becomes possible to see all the TLS handshake messages, that are exchanged between client and server during the TLS handshake process (including information such as cipher suite, session ID, and server certificate). Additionally, the data exchanged between the client and server, including files, emails, and content from web pages, can also be viewed. After decrypting TLS traffic, it is also possible to extract HTTP headers, which contain metadata related to HTTP requests and responses, with info like URL, HTTP method, status code, and cookies.
In this case, where the the analysis is performed over malicious HTTPs traffic with the goal to find the type of malware that has been detected on the system, after the decryption, it is possible to start looking for successful TLS handshakes and HTTP traffic, and this can be done with *tls.handshake.type eq 1 or http.request*. At this stage, the traffic can be analyzed to detect suspicious activities, such as a GET request attempting to reach a ".dll" file. By following the HTTP stream (right-click on the GET → follow → HTTP stream), it becomes clear that the file has already been downloaded, so this file can be saved (file → export object → select the right file) and analyzed using a platform like VirusTotal to determine the type of malware it contains and the potential method of infection. Returning to Wireshark, after further analysis, it is possible to identify suspicious POST requests, such as those directed to a PHP file. This suggests that the infected system is attempting to connect to a command and control server. By right-clicking on the POST request and selecting "Follow" followed by "TLS stream," additional information regarding this connection can be obtained [14].

2.5 Security Information and Event Management (SIEM)

Security information and event management (SIEM) is an approach to security management that combines security information management (SIM) and security event management (SEM) functions into one security management system [15]. The core principles of every SIEM system are to aggregate relevant data from multiple sources, detecting anomalies (such as phishing, ransomware, DDoS, data ex-filtration, insider threats), and take appropriate action. For example, when a potential threat is detected, a SIEM system might log additional information, generate an alert and instruct other security controls to stop an activity's progress. At the most basic level, a SIEM system can be rules-based or employ a statistical correlation engine to make connections between event log entries. Advanced SIEM systems have evolved to include user and entity behavior analytics, as well as security orchestration, automation and response. SIEM systems operate by using a hierarchical network of collection agents to gather security-related events from various sources, which can include network devices such as routers, switches, and wireless access points, as well as servers like web, proxy, and mail servers. They also collect information from security devices including intrusion prevention systems (IPS), intrusion detection systems (IDS), firewalls, antivirus software, and content filters, along with applications running on these devices. Additionally, data is obtained from cloud and SaaS solutions that operate outside of on-premises environments. These collected events are then sent to a centralized management console, where security analysts analyze the data, filter out irrelevant information, identify patterns, and prioritize security incidents for further action. SIEM tools identify and sort the data into categories such as successful and failed logins, malware activity and other likely malicious activity. The SIEM software generates security alerts when it identifies potential security issues. Using a set of predefined rules, organizations can set these alerts as a low or high priority. For example, an user account that, in 25 minutes, generates 25 failed login attempts could be flagged as suspicious, but still be set at a lower priority because the login attempts were probably made by a user who had forgotten their login information. However, if a user account generates 130 failed login attempts within five minutes, it would be flagged as a high-priority event, because it's most likely a brute-force attack in progress.

The importance of SIEM is that it simplifies security management for companies by filtering huge amounts of security data and prioritizing security alerts generated by the software. SIEM software enables organizations to detect incidents that may otherwise go undetected. The software analyzes the log entries to identify signs of malicious activity. In addition, since the system gathers events from different sources across the network, it can re-create the timeline of an attack, enabling an organization to determine the nature of the attack and its effect on the business. A SIEM system can also help an organization meet compliance requirements by automatically generating reports that include all the logged security events among these sources. This automation eliminates the need for manual data collection and report compilation, which would otherwise be time-consuming and prone to errors. Additionally, SIEM systems enhance incident management by allowing the security team to trace the path of an attack across the network, identify compromised sources, and utilize automated tools to prevent ongoing attacks.

Security information and event management solutions provide several key advantages, particularly in enhancing security measures, improving response times, and simplifying the management of complex systems [16]. These systems are designed to increase security effectiveness by enabling analysts to identify and respond to suspicious behavior patterns more quickly and accurately than would be possible when examining data from individual systems. This is important to help prevent successful breaches. Additionally, SIEM solutions simplify the process for IT teams by providing tools that make it easier to monitor and report compliance efforts. By consolidating security event data from multiple applications and devices, SIEM systems significantly reduce the complexity of security management. This consolidation allows for faster and more comprehensive analysis while automating repetitive tasks. Another positive aspect is that companies can use SIEM for a variety of use cases that revolve around data or logs, including security programs, audit and compliance reporting, help desk and network troubleshooting. SIEM also supports large amounts of data so organizations can continue to scale out and add more data.

Despite its advantages, Security Information and Event Management systems come with several limitations. Implementing SIEM can be time-consuming, because the process demands significant support to ensure proper integration with the various hosts within the infrastructure and with

the organization's security controls. The financial investment in SIEM is also substantial, with initial costs potentially reaching hundreds of thousands of dollars. Additionally, ongoing expenses can accumulate, including those for personnel to manage and monitor the SIEM system, as well as annual support fees and the costs associated with software or agents required to collect data. Another limitation is the fact that SIEM tools usually depend on rules to analyze all the recorded data and this can be problematic in environments where networks generate thousands of alerts daily, making it challenging to identify potential attacks due to the sheer volume of irrelevant logs. Moreover, if a SIEM tool is not configured correctly, it may fail to detect critical security events, thus reducing the effectiveness of an organization's information risk management efforts.
When evaluating SIEM products, several critical features should be considered. One of the primary aspects is data aggregation, which involves collecting and monitoring data from various sources. Another key feature is correlation, often integrated into the security event management component of a SIEM tool, which identifies common attributes across different events to reveal patterns of potential threats. The ability to effectively visualize data via dashboards is also critical. These dashboards display aggregated data from multiple sources, including applications, databases, networks, and servers, in a visual format such as charts. This helps identify patterns and ensure that critical events are not overlooked. Additionally, alert mechanisms within SIEM tools are essential to notify users when a security incident is detected. It is also important to consider automation, as some SIEM solutions may include automated functions for security incident analysis and response, thereby improving the efficiency and speed of threat mitigation.
In addition to these features, certain questions should be addressed when assessing the capabilities of a SIEM product. The system's ability to integrate with other security controls must be considered to effectively prevent or disrupt ongoing attacks. The potential for incorporating artificial intelligence (AI), particularly machine learning and deep learning, is also vital for improving the system's accuracy over time. Another factor is support for threat intelligence feeds; it is important to determine whether the system allows the organization to choose preferred feeds or is limited to specific ones. Additionally, the extent of compliance reporting provided by the system, including whether it offers built-in reports for common compliance requirements and the flexibility to customize or create new reports, should be evaluated. Finally, the forensic capabilities of the system are important, particularly its ability to capture detailed information about security events, such as recording packet headers and contents that are relevant to the investigation of incidents.
SIEM solutions are ideal for conducting computer forensic investigations once a security incident occurs. SIEM solutions allow organizations to efficiently collect and analyze log data from all of their digital assets in one place. This gives them the ability to re-create past incidents or analyze new ones to investigate suspicious activity and implement more effective security processes.
The market offers a diverse range of SIEM tools, among which there are Splunk, IBM QRadar SIEM and ELK (Elasticsearch, Logstash, Kibana). Splunk is an on-premises SIEM system that supports security monitoring and offers continuous security monitoring, advanced threat detection, incident investigation and incident response. On the other hand, IBM QRadar is a SIEM platform that provides security monitoring across IT infrastructures. It includes capabilities such as log data collection, threat detection, and event correlation, making it effective in identifying and managing security incidents.

Chapter 3

State-of-the-art

Digital forensics is commonly thought to be confined to digital and computing environments, but in fact, it has a much larger impact on society. Because computers and computerized devices are now used in every aspect of life, digital evidence has become critical to solving many types of crimes and legal issues, both in the digital and in the physical world.
All connected devices generate huge amounts of data that can be used as evidence in investigations and legal proceedings for: data theft and network breaches (to understand how a breach happened and who were the attackers), online fraud and identity theft (to understand the impact of a breach on organizations and their customers), white collar crimes (to collect evidence that can help identify and prosecute crimes like corporate fraud, embezzlement, and extortion) and violent crimes like burglary, assault, and murder (to capture digital evidence from mobile phones, cars, or other devices in the vicinity of the crime).
In the context of an organization, digital forensics can be used to identify and investigate both cybersecurity incidents and physical security incidents. Digital evidence is used as part of the incident response process, to detect that a breach occurred, identify the root cause and threat actors, eradicate the threat, and provide evidence for legal teams and law enforcement authorities. An important thing is to ensure that logs and other digital evidence are kept for a long period and protected in order to avoid unauthorized access, tampering and loss of data.

3.1 Different branches of Digital Forensics

Computer forensics focuses on investigating evidence found in computers and digital storage devices. This field involves examining digital data to identify, preserve, recover, analyze, and present facts and opinions about the information being reviewed. This branch of computer forensics uses similar principles and techniques to data recovery, but includes additional practices and guidelines that create a legal audit trail with a clear chain of custody.
Mobile device forensics primarily aims to recover digital evidence from mobile devices. This field involves investigating any device with internal memory and communication functionalities, including mobile phones, PDA (Personal Digital Assistant, small, mobile, handheld device that provides computing and information storage and retrieval capabilities for personal or business use) devices, tablets, and GPS devices.
Network Forensics involves monitoring, recording, and analyzing network activities. Network data is highly dynamic, even volatile, and once transmitted, it is gone. This means that network forensics is usually a proactive investigation process, which aims to identify potential threats and criminal activities in their early stages, so that timely action can be taken to reduce the damage [17].
Forensic Data Analysis (FDA) refers to the study of digital data and the investigation of cybercrime. FDA may focus on mobile devices, computers, servers and other storage devices, and it typically involves the tracking and analysis of data passing through a network. Investigators employ a range of tools, including decryption and reverse engineering. Some investigators distinguish between "persistent data" stored on a drive and volatile data, which resides in registries, cache

and RAM, and which will be destroyed when the computer is shut down. The goal of Forensic Data Analysis is to detect and analyze patterns of fraudulent activity [18].
Database Forensics is a subfield of digital forensic science concerned with the forensic examination of databases and their metadata. It is the use of electronic data stored in the database to reconstruct the clues, detect crime, and accomplish case cracking. Forensic analysis of a database may involve inspecting and validating the timestamps associated with the update time of a row in a relational table to validate a database user's actions [19].

3.2 Digital Forensic Techniques

Digital forensics involves creating copies of a compromised device and then using various techniques and tools to examine the information. There are different digital forensics techniques that help inspect unallocated disk space and hidden folders for copies of encrypted, damaged, or deleted files [20] [21].

3.2.1 Reverse Steganography

Reverse Steganography is a method that involved analyzing the data hashing found in a specific file. This is done because steganography is technique used by cybercriminals to hide the data in digital files, messages, or data streams. When inspected in a digital file or image, hidden information may not look suspicious. However, hidden information does change the underlying hash or string of data representing the image. To counter this, computer forensics investigators can check and compare the hash values of the original file and the edited file, even if the two files may appear to be similar at first glance the hash values will differ. There are various types of steganography, including text steganography, image steganography, video steganography, audio steganography, and network steganography.

3.2.2 Stochastic Forensics

Stochastic Forensics is a method or technique in the field of computer forensics that helps analyze and reconstruct digital activity that does not produce digital artifacts. A digital artifact can be defined as an unintentional modification of data brought about by doing digital operations. Text files, for example, can be said as digital artifacts that may contain clues or hints related to a digital cybercrime like data theft that modifies file properties. Using stochastic forensics as a technique helps to investigate data breaches that are caused due to insider threats, that might not leave digital evidence behind digital artifacts.

3.2.3 Cross-drive Analysis

Cross-drive Analysis, also known as anomaly detection, is a technique that helps investigators find out the similarities to provide the investigation context using cross-drive analysis. These similarities serve as baselines to detect suspicious events. It typically involves correlating and cross-referencing information across multiple computer drives to find, analyze, and preserve any information relevant to the investigation.

3.2.4 Live Analysis

Live Analysis is a method that is used to examine computers or devices while they are running, it takes place within the operating system using various forensics and system admin tools to get the information from that device or computer utilizing system tools to locate, examine, and retrieve volatile data, usually kept in RAM or cache. The data that is being collected is from the installed software packages, hardware information, etc. Live analysis typically requires keeping the inspected computer in a forensic lab to maintain the chain of evidence properly.

3.2.5 Deleted File Recovery

Deleted File Recovery, also known as data carving or file carving, is a technique that helps recover deleted files. It starts with the process of scanning memory and a computer system for fragments of information that were erased partially in one place but left behind evidence in another area of the examined device. The deleted information can be recovered using forensic tools such as Wise Data Recovery, CrashPlan, etc.

3.3 Digital Forensic Models

Nowadays, as more and more information is stored in a digital format and cyber-crimes have increased, it has become very important and crucial that experts performing digital forensic investigations can conduct analyzes correctly. Various investigation models have been developed since 1984, some of these were for incident response and others were for court admissibility, but all were developed in an attempt to investigate and where necessary prosecute offenders. The goal was to help investigators deal with increasingly sophisticated and complex digital crimes [22].
The method used in conducting a computer forensic investigation directly impacts the results of the investigation, so it is very important to choose the right model to perform an investigation, because a wrong one may lead to incomplete or missing evidence or data.

3.3.1 Computer Forensic Investigative Process (CFIP)

In 1984, the original computer forensics investigation process model was proposed by Mark Pollitt [22]. The model proposed was discussed in Proceeding of the National Information Security Conference and it included four phases: acquisition, identification, evaluation, and admission, which aimed to guarantee the admissibility of evidence in the court.
The first phase is **Acquisition**, where evidence is acquired with approval from authorities and in an acceptable manner. This is followed by the **Identification** phase, where all the evidence is transformed from digital format to a human understandable format. Then there is the **Evaluation** phase that includes tasks that are used to determine whether the component identified in the previous phase is relevant in the case investigated or whether it should be considered as evidence. The last step is the **Admission** phase where all the extracted evidence is presented [23].

3.3.2 Digital Forensic Research Workshop Investigative Model (DFRWS)

The Digital Forensic Research Workshop (DFRWS) model is a collective document created at a Research Workshop organized in Utica USA in 2001. The goal of the workshop was to provide a forum for a new community of academics and practitioners to share their knowledge on digital forensics and it was attended by military, civilian, and law enforcement professionals who use forensic techniques to uncover evidence from digital sources. The group produced a consensus document outlining the state of digital forensics at that time, which included six phases plus an additional step termed Decision. The phases were: Identification, Preservation, Collection, Examination, Analysis, and Presentation.
The first phase is **Identification** and comprehends event or crime detection, resolving signature, anomalous detection, system monitoring, audit analysis, etc. This is followed by the **Preservation** step, in which a proper case management is set, imaging technologies are used and all measurement are taken to ensure an accurate and acceptable chain of custody; Preservation is a crucial principle maintained throughout all forensic phases. The **Collection** takes place directly after the collection of the relevant data according to approved methods, software and hardware; in this step also different recovery and lossless compression techniques are used. Following this step, there are two critical phases: **Examination** and **Analysis**. During these phases, evidence traceability and pattern matching are ensured, hidden data is discovered and extracted, and data mining and timeline analysis are conducted. The final phase of this model is **Presentation**. In this step, tasks include documentation, clarification, creating a mission impact statement, providing recommendations, implementing countermeasures, and offering expert testimony.

The advantages of this model include providing a standard and consistent forensic framework, serving as a foundation for developing other forensic models, and being user-friendly and easily understood by both technical and non-technical users. However, its general nature makes it relatively difficult to test and implement, and it may also seem somewhat rigid [24].

3.3.3 Request for Comments (RFC) 3227

RFC 3227 was proposed in 2002 with the purpose to provide guidelines on the collection and archiving of evidence relevant to a security incident emphasizing its importance and providing a set of recommendations for collecting and protecting information. When collecting evidence, several guiding principles should be followed throughout the process [25]. First, it is important to adhere to the site's Security Policy and to involve the appropriate Incident Handling and Law Enforcement personnel. The system's state should be captured as accurately as possible, with detailed notes including dates and times. The discrepancy between the system clock and UTC should be noted, and the time zone used for each timestamp should be indicated. The notes are important because it is important to be prepared to testify, potentially years later, regarding all actions taken and the exact times they were performed. Changes to the data should be minimized during collection, not only in terms of content but also in avoiding updates to file or directory access times. Changes to the data should be minimized during collection, not only in terms of content but also in avoiding updates to file or directory access times. Special attention should also be paid to all external avenues for data alteration, which should be eliminated. When a choice arises between collection and analysis, priority should be given to collection, with analysis postponed until later. Procedures must be practical and tested for feasibility, particularly during crises. Where possible, procedures should be automated to enhance speed and accuracy, and a methodical approach should be employed. peed is often critical, and if multiple devices need examination, it may be appropriate to divide the workload among team members to collect evidence simultaneously. However, on any single system, evidence collection should proceed step by step, starting from the most volatile data and moving to the less volatile, in accordance with the Order of Volatility. This order of volatility may include, starting with the most volatile: registers and cache, followed by the routing table, ARP cache, process table, kernel statistics, and memory. Next, temporary file systems should be addressed, then disk data, and subsequently, relevant remote logging and monitoring data. The final stages involve documenting the physical configuration and network topology, and lastly, examining archival media.
Since it is too easy to destroy collected evidence, there are some mistakes to avoid. Evidence collection should be fully completed before shutting down the system, as critical information may be lost, and attackers might have modified startup or shutdown scripts to erase evidence. Programs on the compromised system should not be trusted; instead, evidence collection tools should be executed from securely protected devices. Additionally, care must be taken to avoid running programs that modify the access times of files on the system, such as 'tar' or 'xcopy.' Moreover, when removing external avenues for change, simply disconnecting or filtering the network may activate "deadman switches" designed by the attackers to detect network disconnection and subsequently erase evidence.
Privacy considerations must be carefully observed during evidence collection. It is essential to follow the privacy policies of the organization and the legal requirements of the jurisdiction. This involves ensuring that any information gathered alongside the evidence, such as log files or personal data, is not accessed by unauthorized individuals and privacy should not be breached without substantial justification.
There are also some legal considerations regarding digital evidence, as it must meet specific criteria to be acceptable in a court of law. The evidence must be admissible, meaning it must adhere to established legal standards before being presented in court. It must also be authentic, with a clear and verifiable connection to the incident in question. Additionally, the evidence should be complete, providing a comprehensive account of events rather than a selective perspective. The reliability of the evidence is crucial, ensuring that the methods of collection and handling do not introduce any doubts about its authenticity and accuracy. Finally, the evidence must be credible and easily understood by the court.
Collection procedures should be meticulously detailed and they must be clear and designed to

minimize the need for decision-making during the evidence collection process. The methods employed to gather evidence should be both transparent and reproducible. It is important to be able to precisely replicate the methods used and to have these methods evaluated by independent experts to ensure their validity. Whenever possible, generating checksum and cryptographically signing the collected evidence should be considered, as this helps maintain a robust chain of evidence. It is crucial that these actions do not modify the evidence itself.
Evidence must be strictly secured, and the Chain of Custody must be meticulously documented in order to be able to clearly describe how the evidence was found, how it was handled and everything that happened to it. This documentation should include specifics such as who found and collected the evidence, as well as the time and place of these actions. Additionally, it should detail who handled or examined the evidence, when this occurred, and how the evidence was stored. The records must also track who had custody of the evidence during each period and describe how and when custody was transferred, including any relevant shipping numbers and other details.
To store evidence, if possible, it is best to use conventional devices and not some obscure storage system, and access to evidence must be extremely limited and monitored, and in any case always documented.

3.3.4 Abstract Digital Forensics Model (ADFM)

The Abstract Digital Forensics Model (ADFM), proposed by Reith, Carr and Gunsch in 2002, provides a clear and structured way to proceed with particular evidence. It contains nine phase (three more than the DFRWS model) which are Identification, Preparation, Approach Strategy, Preservation, Collection, Examination, Analysis, Presentation and Returning Evidence. The three significant phases introduced in this model were Preparation, Approach Strategy and Returning Evidence.
In this model, the **Identification** phase assumes that the incident type is clearly recognized and determined; this step is crucial because all subsequent steps rely on this initial identification. After this step, there is the **Preparation** phase, where activities include preparing tools, identifying techniques, and securing management support. This step is followed by **Approach Strategy** that was introduced to maximize the acquisition of untainted evidence while minimizing any negative impact on the victim and those nearby. The next phase is **Preservation**, where all acquired data must be isolated and secured to keep them in their actual state. Then there is the **Collection** phase, during which all acquired digital evidence is duplicated and the physical scene is recorded, based on standardized procedures. The next phase is **Examination**, where an in-depth systemic analysis is conducted to find evidence related to the current case. In the **Analysis** phase, the probative value of the examined evidence is determined. The following step is **Presentation** where a summary of the process is developed. The final phase introduced is **Returning Evidence**, which aims to ensure that evidence is either safely returned to its rightful owner or properly disposed of. This model has several advantages, including its versatility in handling a range of digital devices and its accessibility to non-technical observers. Additionally, it has the potential to incorporate non-digital electronic technologies within its framework. However, the model's general nature may present practical challenges, and there is no straightforward or clear method for testing it [24].

3.3.5 Integrated Digital Investigation Process (IDIP)

Integrated Digital Investigation Process (IDIP) was proposed by Carrier & Spafford in 2003, to combine the various available investigative processes into one integrated model [26]. The author introduces the concept of digital crime scene which refers to the virtual environment created by software and hardware where digital evidence of a crime or incident exists. The process begins with the **Readiness** phase, which ensures that the physical and operational infrastructure is prepared to support any future investigations. During this phase, equipment must always be ready for use, and the investigators must be trained to use it effectively. This phase is ongoing throughout the organization's lifecycle and includes two sub-phases: *Operation Readiness* and *Infrastructure Readiness*. Following this step, there is the **Deployment** phase, which provides a mechanism for an incident to be detected and confirmed. This phase consists of two sub-phases: *detection*

& *notification* then *confirmation* & *authorization*. Collecting and analyzing physical evidence are done in the **Physical Crime Scene Investigation** phase and the sub-phases introduced are *Preservation*, *Survey*, *Documentation*, *Search* & *Collection*, *Reconstruction* and *Presentation*. **Digital Crime Scene Investigation** is similar to Physical Crime Scene Investigation with exception that it is now focusing on the digital evidence in digital environment with the use of virtualization. The last one is the **Review** phase, during which the whole investigation processes are reviewed to identify areas of improvement that may results in new procedures or new training requirements.
However, the IDIP model faces some criticisms. Firstly, despite incorporating elements from earlier models, its practicality is questioned. For instance, it portrays the deployment phase, which includes incident confirmation, as independent of the physical and digital investigation phases. In practice, it seems impossible to confirm a digital or computer crime without conducting some preliminary physical and digital investigation. Secondly, the model lacks specificity and fails to clearly distinguish between investigations at the victim's (secondary crime) scene and the suspect's (primary crime) scene, nor does it detail the process of arriving at the latter. Given that a computer can serve as both a tool and a victim, investigations often need to be conducted at both ends to ensure accurate results [24].

3.3.6 Enhanced Integrated Digital Investigation Process (EIDIP)

The Enhanced Digital Investigation Process Model (EDIP), proposed in 2004 by Baryamueeba & Tushaba, seeks to enhance Integrated Digital Investigation Process model by adding two additional steps: Trace back and Dynamite [26].
The investigation process start with **Readiness** phase and the tasks performed are the same as in IDIP. This is followed by the **Deployment** phase, which provides a mechanism to detect and confirm an incident, and it includes both physical and digital crime scene investigations and presentation of findings to legal entities (via Submission phase). It consists of five sub-phases: *Detection & Notification*, *Physical Crime Scene Investigation*, *Digital Crime Scene Investigation*, *Confirmation*, and *Submission*. In the **Traceback** phase, the primary objective is to track down the source crime scene, including identifying the devices and location involved. This phase is supported by two sub-phases: *Digital Crime Scene Investigation* and *Authorization*, which involves obtaining approval to conduct the investigation and access information. Following the Traceback phase, there is the **Dynamite** one. In this phase, investigations are conducted at the primary crime scene with the aim collecting and analyzing the items found in the scene in order to obtain further evidence that the crime originated from there and to identify the potential culprit(s). It consists of 4 sub-phases: *Physical Crime Scene Investigation*, *Digital Crime Scene Investigation*, *Reconstruction* and *Communication*. In the Reconstruction sub-phase, pieces of information collected are put together so as to construct to possible events that could have happened. The Communication sub-phase is similar to the previous Submission phase. The last phase is the **Review** one, where the whole investigation is reviewed and areas of improvement identified.
The model offers several advantages, such as incorporating a wide range of electronic and non-digital technologies and creating a consistent, standardized framework for digital forensic development. This investigative model is also adaptable to future digital technologies. However, it has some disadvantages, including the introduction of additional sub-phases that create ambiguity regarding the activities performed. Additionally, there appears to be duplication of activities, such as Digital Crime Scene Investigation, which is included in the Deployment phase, Traceback phase, and Dynamite phase [24].

3.3.7 Systematic Digital Forensic Investigation Model (SRDFIM)

In the Systematic Digital Forensic Investigation Model the digital forensics investigation process is organized into eleven phases that are: Preparation, Securing the Scene, Survey & Recognition, Documenting the Scene, Communication Shielding, Evidence Collection, Preservation, Examination, Analysis, Presentation and Result & Review [27]. This model was developed with the aim of helping forensic practitioners and organizations for setting up appropriate policies and procedures in a systematic manner and its application is limited to computer frauds and cyber-crimes.

The first step is the **Preparation** phase and it occurs prior to the actual investigation. This phase involves gaining an initial understanding of the nature of the crime and the related activities, preparing materials for collecting and packing evidence, and obtaining the necessary authorizations and approvals. This includes securing search warrants and providing legal notice to relevant parties. Additionally, an appropriate investigative strategy should be developed. Following this, there is the **Securing the Scene** phase, which deals with securing the crime scene form unauthorized access and preserving the evidence from being contaminated. There should be a formal protocol for handing over a crime scene in order to ensure that the chain of custody is properly followed. After this step there is the **Survey & Recognition** phase, where an initial survey is conducted by the investigators for evaluating the crime scene, identifying potential sources of evidence (both devices and people) and formulating an appropriate search plan. The fourth stage is the **Documenting the Scene** phase, which involves proper documentation of both physical and digital crime scenes along with photographing, sketching, and crime-scene mapping. All electronic devices at the scene, along with their power adapters, cables, cradles, and other accessories, must be photographed. If a digital or mobile device is powered on, the current screen display should also be documented and a record of all visible data must be created to help recreate the scene and allow for future review. This is followed by the **Communication Shielding** phase where all the possible communication options of the devices should be blocked, even if the device appears to be in off state, because some communication features like wireless or Bluetooth may be enabled. The evidence are collected in the **Evidence Collection** phase and they include both volatile and non-volatile. While performing this operation, necessary precautionary measures must be taken to ensure the integrity of the evidence. The **Preservation** phase includes copies of digital evidence, packaging, transportation, and storage. Appropriate procedures should be followed and documented to ensure that the electronic evidence collected is not altered or destroyed and all potential sources of evidence should be identified and labeled properly before packing. The next phase is the **Examination**, during which a forensic specialist examines the content of the collected evidence and extracts information to be presented in court. An appropriate number of backups must be created before proceeding to the examination phase, which aims to make the evidence visible while clarifying its originality and significance. Given the large volumes of data collected during the volatile and non-volatile collection phases, it is crucial to convert this data into a manageable size and format to ensure it can be effectively analyzed in the future. The **Analysis** phase is more of a technical review conducted by the investigative team on the basis of the result of the examination of the digital evidence. The key activities during this phase include identifying relationships between data fragments, analyzing hidden data, assessing the significance of information obtained, reconstructing event data from the extracted information, and drawing accurate conclusions. According to the guidelines from the National Institute of Justice (2004), this phase should include time frame analysis, hidden data analysis, application analysis, and file analysis of the extracted data. The results from the analysis phase may reveal the need for further extraction and analysis steps. It is essential to verify the consistency of the chain of evidence and the timeline of events, and using a combination of analytical tools is recommended to achieve the best results. Additionally, the outcomes of the analysis should be thoroughly and accurately documented. Following this, there is the **Presentation** phase that involves preparing a report that provides a detailed summary of the steps taken during the investigation and the conclusions reached. This report is then presented to the relevant authorities, such as a court of law in the case of a criminal investigation, or to corporate management in the case of an incident. The last phase is the **Result & Review** one, the whole investigation is reviewed (step by step) and areas of improvement identified.

The model offers several advantages, such as recognizing the need for interaction between the investigator and various resources, which can lead to better-defined case goals and effective exploratory testing. Additionally, it supports capturing investigative expertise for developing advanced tools, including automated digital evidence collection. Another key advantage is its comprehensive scope, incorporating most significant activities from existing models and aligning with forensic laws and National Institute of Justice guidelines. It effectively addresses the critical issue of collecting digital evidence from both volatile data and live responses, which is essential for cybercrime investigations and prosecution [24].

However, the model's clarity on how it applies to different crimes could be better, its general applicability is not clear until it is used in the context of a specific crime, making the process

details ambiguous. It focuses heavily on the technical side (examination and analysis). The model revealed some similarities in some of the phases which could be regrouped to make it more coherent and improve clarity (for example, Survey and Recognition could be part of Preparation, Documenting the Crime Scene and Communication Shielding could also be part of Securing the Crime Scene, since these two independent phases in this model in reality are part of Securing Crime Scene). Also, it lacks coverage of all cybercrime investigation aspects, mainly focusing on obtaining digital evidence from a standalone machine rather than a networked or distributed system. This limitation restricts its usefulness in more complex cybercrime scenarios where tracing digital footprints across multiple systems is essential.

3.3.8 ISO/IEC 27037

ISO/IEC 27037 was first published in 2012 by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission), and it is an internationally recognized standard that offers comprehensive recommendations for the effective management of electronic evidence. The standard contains guidelines for the identification, collection, acquisition, conservation and transport of digital evidence in order to facilitate its exchange between multiple countries using common methodological protocols. These processes are required in an investigation that aims to maintain the integrity of digital evidence. It applies universally across various domains (civil, criminal, or extrajudicial) without referring to specific systems or legal regulations and ensures that responsible individuals manage digital evidence in practical ways that are acceptable worldwide. This standard provides general guidelines for the collection of non-digital evidence that may be useful in the analysis stage it intends to provide guidance for individuals dealing with potential digital evidence and they include e Digital Evidence First Responders (DEFRs), Digital Evidence Specialists (DESs), incident response specialists and forensic laboratory managers.
Given the fragile nature of digital evidence, it is crucial to implement a robust methodology to ensure the integrity and authenticity of potential digital evidence. This International Standard does not address the methodology for legal proceedings, disciplinary actions, or other related procedures that fall outside the scope of identifying, collecting, acquiring, and preserving digital evidence. When applying this international standard, which is intended to serve as a practical guideline for DEFR experts or DES in investigations involving digital evidence, it is mandatory to comply with national laws, rules and regulations. his Standard does not cover the analysis of digital evidence or replace jurisdiction-specific requirements regarding admissibility, evidential weight, relevance, and other judicial limitations on the use of digital evidence in court. It may, however, assist in the exchange of digital evidence between jurisdictions. To maintain the integrity of digital evidence, users of this International Standard must adapt and modify the procedures according to the legal requirements of the relevant jurisdiction. In most jurisdictions and organizations, digital evidence is governed by three core principles: relevance, reliability, and sufficiency. Digital evidence is considered *relevant* when it contributes to proving or disproving an aspect of the case under investigation. While the specific definition of *reliable* may differ across jurisdictions, the general meaning of the principle, that digital evidence must be verified as authentic and accurately represent what it claims to be, is widely accepted. It is not always necessary for the DEFR team to collect all available data or make a complete copy of the original digital evidence. In many jurisdictions, the principle of *sufficiency* requires that the DEFR collect enough potential digital evidence to fully examine or investigate the key elements of the matter.
All processes used by the Digital Evidence Forensics and Response team and Digital Evidence Specialists must be validated before implementation. The DEFR or DES should also ensure that all actions are thoroughly documented, establish and apply a method to verify the accuracy and reliability of the potential digital evidence copy against the original source, and acknowledge that preserving potential digital evidence may sometimes involve intrusive actions.
There are four crucial aspects to handling digital evidence: auditability, justifiability, and either repeatability or reproducibility, depending on the circumstances. *Auditability* ensures that an independent assessor or other authorized parties can review and evaluate the actions taken by a DEFR team or DES. This is achieved by documenting all actions. *Repeatability* is demonstrated when identical test results are obtained under the same conditions, using the same measurement procedures, methods, and instruments. These results should be replicable at any time after the

original test. A skilled and experienced DEFR should be able to follow the documented procedures independently and achieve the same results, without requiring additional guidance or interpretation. *Reproducibility*, on the other hand, is confirmed when the same test results are obtained using the same measurement method, but with different instruments and under varying conditions. These results should also be reproducible at any time after the original test. The need to reproduce results varies based on jurisdiction and specific circumstances, so the DEFR or the individual conducting the reproduction must be aware of the relevant conditions. The DEFR must be able to justify all actions and methods used in handling potential digital evidence. *Justification* is achieved by demonstrating that the chosen approach was the most effective for obtaining all potential digital evidence.
Although the full process of handling digital evidence includes additional activities such as presentation and disposal, this International Standard focuses only on the initial handling stages, which encompass the identification, collection, acquisition, and preservation of potential digital evidence. The **identification** process involves finding, recognizing, and documenting potential digital evidence. It should specifically focus on identifying digital storage media and processing devices that may hold evidence relevant to the incident. The volatility of the data must be assessed to ensure that the collection and acquisition processes are conducted in the correct order, minimizing any potential damage to the evidence and ensuring the best possible preservation. Additionally, this process should consider the possibility of hidden digital evidence. Once the digital devices that may contain potential digital evidence are identified, the DEFR and DES should decide whether to collect or acquire during the next process. *Collection* is a critical step in the digital evidence handling process, where devices that may contain potential digital evidence are relocated from their original site to a laboratory or other controlled environment for subsequent acquisition and analysis. This process involves documenting each step, including the packaging of devices before transport. It is essential for the Digital Evidence Forensics and Response (DEFR) team and Digital Evidence Specialists (DES) to collect all materials that could be related to the potential digital evidence. Without careful handling, there is a risk that the evidence may be lost or damaged. The **acquisition** process involves creating a copy of the digital evidence and documenting the methods and activities involved. The methods used to acquire potential digital evidence should be, as far as practically possible, reproducible or verifiable by a qualified DEFR professional. The acquisition should be performed in the least intrusive way to minimize any potential changes to the evidence. The method must generate a reliable copy of the digital evidence or digital devices that may hold relevant information. Both the original source and the copied evidence should be verified using a reliable verification function, which must be accurate and acceptable to the person who will use the evidence. In some cases, creating a complete digital copy of an evidence source may not be practical or permissible, especially if the source is too large. In such situations, a DEFR professional may conduct a logical acquisition, focusing on specific types of data, directories, or locations. This process typically targets data at the file and partition levels. To ensure its effectiveness in the investigation, potential digital evidence must be preserved, maintaining its integrity throughout the process. **Preservation** should begin as soon as digital devices containing potential evidence are identified and continue throughout the entire handling process. The DEFR team must be able to prove that the evidence has remained unaltered since its collection or acquisition. If any changes are unavoidable, the DEFR should provide a clear explanation and document the actions taken.
In any investigation, the Digital Evidence Forensics and Response team must be able to account for all data and devices while they are in their custody. The *chain of custody* record is a document that tracks the movement and handling of potential digital evidence. This record should be established starting from the collection or acquisition of the evidence. It involves documenting the history of the item from when it was first identified and collected by the investigation team to its current status and location. Maintaining a chain of custody record is essential for identifying who has had access to the evidence and how it has been handled over time.

3.3.9 NIST IR 8354

NIST IR 8354 was published in 2022 and it is an assessment of the current scientific foundations of digital forensics. The National Institute of Standards and Technology (NIST), which is part of the U.S [28]. Department of Commerce, conducted this scientific foundation review to document

and consolidate the information that supports forensic analysis methods. The review also aims to identify any existing knowledge gaps in these methods. This assessment was led by a senior computer scientist and a multidisciplinary team from various areas at NIST. This document focuses on techniques for analyzing digital data stored in mobile device memory, computer memory, or secondary storage within an active computer. Secondary storage comprehends devices like hard drives, flash drives, removable drives, and external storage media such as CDs, DVDs, and memory cards. The document evaluates the validity of these techniques, but it does not address how effectively they are applied in practice, best practices for their implementation, or any legal restrictions imposed by the courts.

In digital investigations, tasks can be organized in various ways. For the purposes of this report, tasks are grouped into several key categories. These include protecting original data from unintended modification, acquiring digital data, and ensuring the integrity of the acquired data. Additionally, the process includes recovering deleted data, navigating the acquired digital data, identifying and extracting data artifacts, and analyzing the extracted artifacts.

Although **data protection** is a crucial part of the acquisition process, it is typically addressed as a separate issue due to its importance. Data can be acquired in various ways, depending on whether it is directly accessible from a storage device like a hard drive or flash drive, or whether it requires more complex methods for cloud storage, mobile device memory, or computer memory. Before data can be copied directly from a storage device, the device must be connected to a computer to access and duplicate the data. This connection may pose a risk if the computer alters the contents of the device before the copy is made. To mitigate this risk, a device known as a write blocker is often used to monitor the connection and prevent any commands that might modify the contents of the device. However, write blockers do not entirely eliminate the risk of changes, as some alterations can be triggered by the storage device itself. The **acquisition of digital data** is a core component of digital forensics, primarily involving the creation of a copy of the data for examination. This process is typically reliable and straightforward, with error-correcting codes used by computers to ensure that copies are complete and accurate. The copied data are often saved in a container file that represents the acquired data, although sometimes the data are copied directly from the source device to another device. There are currently over 30 different container file formats available for storing digital data. Among the most commonly used formats are raw images (dd format) and e01 (Expert Witness). There is a recognized need for a standard format, and the Advanced Forensic Format has been proposed as a standard and is available from some digital forensic tool vendors. Once digital data have been acquired into an image file, it is important to implement measures that will allow for future **data integrity verification** to ensure that the image file has not been altered. This can be done with cryptographic hashing, which is a reliable method employed in high-security contexts to detect both accidental and intentional modifications. A cryptographic hashing algorithm must meet several key requirements. It should compute hash values quickly and be resistant to collisions, meaning it is computationally infeasible to find two different files with the same hash value. This feature is crucial because it prevents the possibility of replacing the original data with modified data that has the same hash. Additionally, the original data cannot be reconstructed from the hash value. Any alteration to the original file will result in a change in the hash value; for example, modifying even a single byte in the original file will typically alter about half of the bytes in the resulting hash output. Most operating systems do not immediately overwrite deleted data, which means that such data can often be recovered, at least partially. When a storage device has had multiple owners, it is possible to retrieve data from previous users, not just the current owner. This situation can sometimes reveal incriminating evidence unrelated to the current owner of the storage device. Three common techniques are used to **recover deleted data** files from storage locations that have been marked as available for new allocations. *Metadata-based file recovery* takes advantage of the fact that file systems often do not immediately erase or overwrite deleted data; instead, they simply mark the data as deleted and make the storage space available for reuse. This metadata can be used to locate the deleted data. *File carving* involves identifying deleted files by searching for specific data patterns that signify the beginning and end of files generated by certain applications. This method is used when metadata is not available to assist in the recovery process. *Deleted record recovery* focuses on applications such as databases (e.g., MySQL, SQLite) or the Windows Registry, which keep records that might be marked as deleted but not yet overwritten. Recovery tools analyze the internal structure of application files

to find and recover these deleted records. As applications continuously add, update, and delete records, this method leverages the application's design to recover data that has not been fully erased. After data have been acquired, the examiner must analyze the data using specialized tools. These tools typically present the data in a manner similar to how it appeared in its original environment. They need to be able to recognize and interpret (**parse**) the data structures and metadata within the acquired data to **navigate** the file system and display its contents. However, forensic tools may not always support every file system encountered. When new file systems are introduced, there is often a delay before forensic tools can accommodate them. If a tool encounters an unsupported file system, it generally treats it as unallocated space. After parsing and navigating the data, the next step is to identify items of interest. An examiner typically uses an iterative approach to address questions that emerge during an investigation. Building a narrative to describe the events of interest or answer investigative questions involves **identifying, finding, and extracting relevant artifacts**. When a particular question arises, the examiner selects a specific artifact for closer examination, locates it within the data, and then extracts it for further analysis. This can be achieved using several methods. For instance, keyword searches can identify files containing specific strings, while document retrieval methods locate files related to particular topics. Metadata attribute matching helps find files based on criteria such as the date a file was last updated. Additionally, matching file properties, such as a cryptographic hash of known contraband, can also be used to identify relevant files. The **analysis** phase involves evaluating the **results**, with several important factors to consider. This includes assessing and considering alternative hypotheses and verifying that the elements of the developed narrative are consistent. Any contradictions that arise during this process must be carefully noted. There are different classes of analysis tools that can help an examiner understand the case data. For instance, a timeline tool can organize events in chronological order, offering an overview of their relationships. Link analysis tools help identify connections between entities in an investigation, such as patterns of communication between individuals. Additionally, Artificial Intelligence (AI) tools, employing techniques like deep learning, are capable of improving performance over time. These tools can uncover hidden relationships between case elements or search through data to identify relevant information.

3.3.10 Comparison between models

Steps	CFIP	DFRWS	ADFM	IDIP	EIDIP	SRDFIM
Acquisition	✓					
Identification	✓	✓	✓			
Preparation			✓			✓
Approach Strategy			✓			
Operation Readiness				✓	✓	
Infrastructure Readiness				✓	✓	
Detection & Notification				✓	✓	
Confirmation & Authorization				✓	✓	
Securing the Scene						✓
Preservation		✓	✓	✓	✓	✓
Survey				✓	✓	
Survey & Recognition						✓
Documentation				✓	✓	✓
Communication Shielding						✓
Evaluation	✓					
Collection		✓	✓			✓
Search & Collection				✓	✓	
Confirmation					✓	
Submission					✓	
Examination		✓	✓			✓
Analysis		✓	✓			✓
Reconstruction				✓	✓	
Communication					✓	
Admission	✓					
Presentation		✓	✓	✓	✓	✓
Review				✓	✓	
Result & Review						✓
Returning Evidence			✓			

Steps	RFC 3227	ISO/IEC 27037
Acquisition		✓
Identification		✓
Preservation		✓
Collection	✓	✓
Archiving	✓	

Chapter 4

Use Cases

The use cases taken as reference are five and they try to simulate possible IT security incidents inside companies, in order to recreate and then analyze them, with the aim of deriving approach strategies to manage these situations.
A common procedure to all these cases is to form a forensic team, preserve all possible evidence, isolate the affected systems in order to avoid the expansion of the problem and subsequently acquire forensic images of the affected devices so that the investigators have an exact copy to work on.

4.1 RansomExx (Ransomware)

This case involves a ransomware attack that targeted a company and compromised its security, leading to the theft of sensitive data and a ransom demand to regain possession of it.
The access to the network was obtained via malicious links sent through email to the employees. Once the links were opened from the company computers, the ransomware gained access to the network and spread until it reached its target.
In case of this type of incident, the forensic team focuses on scanning network connections, event logs, RAM and other volatile data in order to obtain relevant information about the attack.
The forensic team analyzes both firewall logs, from which it can obtain important information regarding movement within the network (such as source and destination IP addresses, ports, protocols and type of traffic), and data collected by SIEM tools, which can come from servers, network devices and applications and are used by the tool to identify possible threats and are therefore a possible source of relevant information for the continuation of the investigation.
It is also important to analyze the emails to identify the one containing the malicious link that brought the ransomware into the system and then try to trace the identity of the attacker.
According to cyberint, 2023 is considered the most successful year for ransomware groups in history. In 2022, a total of 2,809 ransomware attacks were recorded globally. By 2023, this number had surged significantly to 5,070 attacks, indicating an increase of over 55%. The second and third quarter (Q2 and Q1) alone claimed more victims than the entire 2022, with 2903 victims [29]. While the number of ransomware attacks decreased in Q1 2024 to 1,048 cases, in Q2 2024, it increased to 1,277 cases. This is almost a 21.5% increase compared to Q1 2024. When examining the top 10 countries most impacted by ransomware attacks, the United States maintains its leading position, accounting for approximately 49.8% (2,175) of all incidents. Following the United States there are the United Kingdom (286), Canada (198), Germany (158), France (140), Italy (134), Australia (107), Spain (78), Brazil (68), and India (61) [30].
In October 2023, the Toronto Library system was hit by a ransomware attack that took down their entire service. The library refused to pay the ransom, but since they had no backups, it took weeks to restore limited in-library services, and the online library was fully active again after more than four months. A forensic analysis conducted by third-party experts led the library to conclude that the attackers, in this case, had exploited a vulnerability in an internet-facing server, allowing them to exfiltrate and encrypt data from a file server. Forensic investigators also found

that the hackers had stolen the personal information of current and former staff members, dating back to 1998: their names, social insurance numbers, dates of birth, home addresses, and copies of government-issued ID they gave to the library [31].

4.2 Data Breach

This case involves a data breach that can be caused by a vulnerability in the software code, which was exploited to obtain and steal, for example, patient data from the Community Health Systems, or caused by the collaboration between an external attacker and an employee who provides him with information regarding a project.
In case of a Data Breach, it is possible to obtain important information, such as the location and the time of the incident, from the analysis of security events recorded by the Intrusion Detection System (IDS) and the Intrusion Prevention System (IPS).
An important step is to determine whether the breach was caused by a system vulnerability or by an human error, so that a deeper examination of the root of the problem can be conducted in order to find a solution.
An example of data breach is the one that hit Twitter (now known as X) where more than 220 million users' email addresses were leaked. In this case, although it appears that no personal information other than email addresses was compromised, the incident poses significant privacy risks, as many people can be easily identified by their email address, particularly if they use the their name or the name of their business. The stolen information dates back to 2021, when cybercriminals discovered a flaw in Twitter's systems. This API vulnerability allowed them to input email addresses or phone numbers to verify their association with a Twitter ID. The perpetrators used web scraping software to search the Internet for publicly available email addresses linked to Twitter accounts. Consequently, the hackers were able to correlate the email addresses and IDs of hundreds of millions of Twitter users [32].
Another case is the one which involved the Community Health Systems, a major US hospital operator, which suffered a data breach that exposed sensitive information of its patients and employees. The breach compromised data of approximately 4.5 million users and the data exposed included social security numbers, names, addresses, birth dates, telephone numbers, and other personal identification information. The Community Health Systems was hacked by an unauthorized individual who exploited a previously unknown vulnerability in Fortra's GoAnywhere MFT platform, a secure managed file transfer software solution designed to streamline data exchange between systems, employees, customers, and trading partners [33].

4.3 Spear Phishing

This is the case of a spear phishing attack in which the targets were some executives of a company present in the consumer electronics, communications and financial services sectors.
The incident, caused by emails containing malicious links sent to specific users within the organization, led to the company's security being compromised with the aim of obtaining unauthorized access and sensitive data. In these cases, the emails used to try to deceive the recipients contain personal information in order to appear as genuine as possible.
In case of an attack of this type, it is important to also focus on analyzing user behavior to identify suspicious activities and dedicate some time to analyzing the victim's emails in order to try to understand when, how and by who the attack was performed.
To analyze the user behavior, the forensic team can use the data collected by the User and Entity Behavior Analytics (UEBA) system which concerns the activities of users and entities and which are collected from system logs.
It is also important that the forensic team checks if there are any malware installed which may come from malicious links present in emails.
Cybercriminals continue to target organizations with spear-phishing attacks, posing significant challenges for many companies. According to Barracuda market research, 50% of surveyed organizations fell victim to spear phishing in 2022, and 24% experienced at least one email account compromised through account takeover [34]. The new report on 2023 spear-phishing trends presents

proprietary data and analysis, based on a dataset including 50 billion emails across 3.5 million mailboxes, which includes nearly 30 million spear-phishing emails. Although spear-phishing attacks are low in volume, they are widespread and more successful compared to other types of email attacks. In 2022, 50% of the organizations analyzed were victims of spear phishing, with a typical organization receiving five highly personalized spear-phishing emails per day. According to Barracuda data, spear-phishing attacks constitute only 0.1% of all email-based attacks but account for 66% of all breaches. Organizations are facing various impacts from successful spear-phishing attacks and struggle with timely detection and response. Among those that experienced such attacks, 55% reported machines are infected with malware or viruses, 49% had sensitive data stolen, 48% had login credentials stolen, and 39% reported direct monetary loss. On average, it takes organizations nearly 100 hours to identify, respond to, and remediate a post-delivery email threat, with 43 hours spent detecting the attack and 56 hours on response and remediation.
In 2014, Xoom experienced a spear-phishing attack involving CEO fraud, where attackers impersonated a CEO or another senior executive in a fake email targeted at employees with the authority to perform specific operations. The company admitted that its financial reports in the first quarter of 2015 would highlight a one-off charge as a consequence, resulting in losses of over $30 million. This attack happened because fraudsters impersonated an undisclosed senior employee requesting a transfer, and the accounts department made this transfer [35].

4.4 Intellectual Property Theft

In this case, an employee of a Biotech Company left the organization and took with him trade secrets relating to the production of a drug from his previous company to the new competitor company.
In case of an intellectual property theft, the forensic analysis carried out by the team must go beyond simply determining the movements of users and the contents of communications, but must dig deeper into the data to obtain a more detailed report of what exactly happened during the crime. The goal is to determine whether data was saved or deleted, to investigate the use of USB sticks, including the specifics of when and where they were used, and to determine if programs were used to cover tracks. Additionally, it aims to uncover whether an email account or cloud-based storage was utilized, if emails were sent to competitors, and if file transfers to a home computer occurred via remote access.
Each year, the National Intellectual Property Rights Coordination Center (IPRCC) reports on enforcement actions taken by the US government against intellectual property theft, counterfeit goods, and fraud. In the last year, the IPRCC has seen a substantial increase in enforcement against these crimes. Specifically, cases initiated against intellectual property theft have risen by 21%, criminal arrests by 39%, indictments by 99%, and convictions by 29%. Although seizure incidents have slightly declined by 9%, the total estimated cost of stolen American intellectual property has increased from $822.3 million to $1.12 billion, marking a 36% rise [36].
According to Verizon's 2023 Data Breach Investigations Report, insiders may steal data for financial benefit, espionage purposes, ideological reasons, or because of a grudge, and for organizations, insider data theft may cause financial losses, reputational damage, loss of customer trust, and legal liabilities [37].
In May 2023, two former employees stole and leaked Tesla's confidential data, including personally identifiable information on over 75,000 employees, to the German news outlet Handelsblatt. An investigation revealed that malicious actors had breached the company's IT security and data protection policies to unlawfully obtain and disclose 23,000 internal documents from Tesla, amounting to nearly 100 gigabytes of confidential information [38].
In May 2022, Apple initiated a lawsuit against Rivos, a chip development startup, alleging the theft of trade secrets following the hiring of over 40 former Apple employees by Rivos. Apple claimed that at least two of these ex-employees took gigabytes of confidential information before transitioning to Rivos. According to Apple, Rivos recruited these former employees to develop competing system-on-chip (SoC) technology. Apple has invested billions of dollars and over a decade of research in creating the SoC designs currently used in iPhones, iPads, and MacBooks. Possession of these trade secrets would have significantly benefited Rivos in its competition with Apple [37].

From these situations it is possible to understand how to try to avoid and react to these types of incidents. It is important to ensure that access to sensitive data is limited, also considering implementing the principle of least privilege. Another important feature is the use of robust user activity monitoring and user and entity behavior analytics (UEBA) tools to reinforce the protection of the organization's intellectual property.

4.5 Advanced Persistent Threat

An Advanced Persistent Threat (APT) attack refers to a sophisticated cyber attack orchestrated by highly skilled and motivated adversaries, using stealthy techniques to gain unauthorized access to sensitive systems and maintain persistence within the targeted network for an extended period of time. The primary objective is often to steal sensitive data or sabotage critical organizational infrastructure.
The forensics team uses Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) to identify potential Indicators of Compromise (IOC) and anomalous behavior within the network; this helps understand how many systems, networks or users are affected and what type of malware, tools or techniques the attackers are using. One important thing is to look for signs of lateral movement, privilege escalation, or data exfiltration, which can indicate the attackers' goals and capabilities.
The forensic team analyzes firewall logs which can provide valuable information on the presence of malware or APTs within the network; these threats often use sophisticated techniques to avoid detection, making identification through traditional security measures more difficult.
By analyzing firewall logs, the forensics team can look for signs of malicious activity, such as suspicious connections to known command-and-control servers or repeated failed login attempts. An in-depth analysis can help organizations identify and respond to these threats promptly, minimizing the potential damage.
Seqrite, an Enterprise Security brand, reported a series of cyberattacks targeting micro, small, and medium-sized businesses in India, attributed to the Gorgon group (also known as Subaat), a threat actor believed to be associated with Pakistan-based interests. It is important to note that these small and medium-sized businesses, employing approximately 40% of India's workforce, form the backbone of the country's economy and account for nearly 45% of India's manufacturing industry. The attackers used COVID-related themes to trick victims into opening malicious documents, such as a file named *face_mask_order.zip*, which exploited the CVE-2017-11882 vulnerability to execute arbitrary code on the computer. The final payload deployed was Agent Tesla [39].
Another example of Advanced Persistent Threat attacks is the SolarWinds one, which was a significant supply chain attack attributed to APT29 (Cozy Bear), a Russian-state-sponsored APT group [40]. The attackers compromised the SolarWinds Orion software platform, used by thousands of organizations for IT infrastructure management. In the Orion hack, a backdoor was created, by malicious pieces of code into the Orion framework, which could be accessed by the hackers to impersonate accounts and users of victim organizations. This backdoor allowed the hackers to access system files and hide their tracks by blending into the Orion activity, masking the malicious code from antivirus packages. This enabled the threat actors to infiltrate the networks of multiple high-profile targets, including U.S. government agencies and Fortune 500 companies [41].

Chapter 5

Docker Setup for Attack Simulation

In this thesis, simulations were conducted to recreate real-world scenarios of cyber attacks against companies, followed by an analysis of the impacted environment using forensic tools. Docker, which is an open platform for developing, deploying, and running applications that offers the ability to package and run an application in a loosely isolated environment called a container, was used to achieve these goals.
Isolation and security enable multiple containers to run simultaneously on a single host. Containers are also lightweight and include all necessary components to run an application independently, eliminating the need to rely on the host's installed software.
An important, and also useful, aspect of Docker is that it provides flexibility in managing the various containers that are part of the virtualized environment, so it is possible to make changes to the basic configuration used as a starting point for different real-world scenarios simulations.
In this case, Docker containers represent the various components of an architecture that simulates a typical company structure. This architecture is divided into three parts: the demilitarized zone (DMZ), which hosts the servers, including the database and email servers; the internal zone, containing all company devices, such as computers and workstations; and finally, the external network, representing the internet and any elements outside the company's boundaries.

5.1 Containers Creation

In the context of recreating a company-like environment using Docker, each container is composed of a Dockerfile and a bash script. The **Dockerfile** serves as a blueprint for the container, specifying the base image, which in this case is the latest version of Ubuntu, and outlining the necessary packages to be installed, such as telnet and various networking tools. This file also includes instructions to copy the setup.sh script from the computer of the host into the container and to make it executable.

Listing 5.1: Dockerfile Example

```
FROM ubuntu:latest

RUN apt-get update && \
    apt-get install -y telnet nano net-tools traceroute && \
    apt-get install -y iputils-ping iproute2 iperf sudo faketime

COPY setup.sh /usr/local/bin/setup.sh

# Make the script executable
RUN chmod +x /usr/local/bin/setup.sh

# Run the script
CMD ["/bin/bash", "-c", "/usr/local/bin/setup.sh"]
```

The **setup.sh** file is a bash script that plays a critical role in configuring the network settings of the container, including defining routing paths and setting up DNS servers, ensuring that the container can communicate effectively within the simulated network environment. At the end of the script there is a command, *tail -f /dev/null*, which is very important because it maintains the container alive, so it is possible to interact with it.

Listing 5.2: Setup file

```
#!/bin/bash

# Routing
ip route del default via 192.168.1.254
route add default gw 192.168.1.1

# Set up DNS
echo "nameserver 8.8.8.8" > /etc/resolv.conf
echo "nameserver 8.8.4.4" >> /etc/resolv.conf

# Keep the container running
tail -f /dev/null
```

5.2 Docker Compose

Docker Compose is a tool that expands Docker with support for multi-container management. Using Docker Compose provides several benefits that simplify the development, deployment, and management of containerized applications. It allows applications with multiple containers to be defined and managed within a single *YAML* file, simplifying the coordination of services and enabling easy management and replication of the application environment. This setup allows all containers to be started or stopped with a single command. Additionally, Compose caches the configurations used to create containers, so when a service is restarted without changes, it reuses the existing containers. This feature allows for quick modifications to the environment, facilitating faster application development [42].
This is a portion of the *docker-compose.yml* file, where the **services** section defines individual services that will be deployed in containers:

Figure 5.1: Docker Compose example.

```yaml
services:
    email_server:
        build: ./dmz/email_server
        container_name: email_server
        privileged: true
        ports:
          - "587:587" # SMTP over SSL/TLS (SMTPS) with STARTTLS
          - "993:993" # IMAP over SSL/TLS (IMAPS)
        volumes:
          - ./maildata:/var/mail
        networks:
          dmz_net:
            ipv4_address: 192.168.1.4
        cap_add:
          - NET_ADMIN
        depends_on:
          - firewall
```

In this case, the service is the email server. The **build** option defines where Docker should look for the *Dockerfile* to build the container image for the email server. Then there is the **container_name** that assigns a specific name to the container, and this provides a straightforward way to identify and manage the containers, especially in multi-container setups.
The next option is **privileged** and it grants elevated permissions to the container, similar to root access on a host machine. This elevated access might be necessary for the email server to perform low-level networking tasks, such as managing network interfaces or handling port forwarding.
Port management can also be configured using the **ports** option, which allows one or more mappings between the host and the container. For instance, in this configuration, port 587 on the host is mapped to port 587 in the container to enable SMTPS. It is not necessary, however, for the port numbers to match; for example, a mapping could specify 8080:80 if different host and container ports are required.
The **volumes** section enables persistent storage for containerized applications: the email server stores user emails and related data in */var/mail* within the container and, by mapping this to *./maildata* on the host, email data persists even if the container is stopped or recreated, preventing data loss and allowing seamless backups or data migrations.
There is also the possibility to assign a static IP address within a custom network, allowing for controlled and secure communication between containers. This can be done using the **networks** option. While only one network is specified in this case, multiple networks can be added if needed.
The next configuration parameter in this example is **cap_add** and it is used to grant additional permissions to the containers, which are typically restricted in their capabilities for security purposes. By adding NET_ADMIN, the container gains the ability to modify network settings, such as adjusting firewall rules or configuring network interfaces.
The last option, **depends_on** defines a startup dependency, instructing Docker to start the firewall service before the email_server. It is worth noting that depends_on does not wait for the

firewall service to be fully ready, only that it is started first.
The *docker-compose.yml* file also allows for the definition of multiple networks within the Docker environment. This can be configured within the **networks** section.

```
networks:
  dmz_net:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.1.0/24
          gateway: 192.168.1.254
```

In this example, the created network is **dmz_net**, which represents a demilitarized zone (DMZ) designated for servers, such as the email server.
The initial parameter specified is the network **driver**. Here, the driver is set to *bridge*, which creates an isolated network for the containers, allowing each container connected to dmz_net to communicate with others on the same network. Alternatively, using *host* as the driver parameter removes network isolation between the container and the host. The *overlay* driver connects multiple Docker daemons together and enables Swarm services and containers to communicate across nodes without the need for OS-level routing. The *ipvlan* driver provides to the users full control over IPv4 and IPv6 addressing, while the *macvlan* driver assigns each container a unique MAC address, making it function as if it were a physical device on the network. The Docker daemon then routes traffic to containers based on their MAC addresses. Finally, the *none* driver completely isolates the container from both the host and other containers [43].
In the end, there is the **ipam** (IP Address Management), where the subnet and gateway are defined.

5.2.1 Operations in Docker

In Docker there are several useful commands to manage and interact with the containerized applications. The command **docker-compose up –build** is used to start and rebuild services defined in a Docker Compose file, ensuring containers are based on the latest configuration and code changes, which is particularly useful during iterative development. Then there is the **docker-compose down –volumes –remove-orphans** command that is utilized to stop containers and remove containers, networks, volumes, and images created by the *up* command [44]. This command is useful because not only stops and removes running containers but also cleans up any associated volumes and orphaned containers that are no longer defined in the compose file. This approach helps maintain a clean environment and prevent the accumulation of unused resources. For an overview of all containers, whether active or inactive, **docker ps -a** displays detailed information on each, enabling effective monitoring and management. As disk space can become limited due to unused images, containers, and networks, the **docker system prune -a** command performs a cleanup, freeing up space by removing all resources not actively in use.
Another important and useful command is **docker exec -it container_name_or_id bash**, which is a powerful tool for interacting directly with a running container's environment. By using this command, Docker allows users to open an interactive terminal session (*-it*), providing real-time access to a container. The *bash* portion of the command specifies that the session will open with the Bash shell, provided the container's image includes Bash; this grants users access to a familiar command-line interface within the container. It is also possible to insert, instead of bash, a command to execute without opening the command-line interface.
In certain scenarios, such as digital forensics, where the exact state of a container needs to be preserved as an image, the **docker commit container_id commit_name** command captures the container state into a new image, providing a way to save and replicate unique configurations. To save an image created from a container commit, the **docker save -o /path/to/dest/folder/image.tar commit_name** command is used. This process generates an image.tar file containing the container image at the specific state of the commit. This approach is particularly useful, as in this thesis, for extracting images to be further analyzed with forensic tools.

5.3 Email Server

The email server is placed in the dmz and was built using postfix and dovecot. It is a local email server created to allow the exchange of emails between employees and also with people outside the company.
Postfix is a hugely-popular Mail Transfer Agent (MTA) designed to determine routes and send emails. In this case, Postfix uses port 587 for SMTP (Simple Mail Transfer Protocol) submission. This port is designated for sending outgoing emails from email clients to the mail server and is meant specifically for authenticated SMTP connections. The 587 port support STARTTLS, TLS encryption, which provides a secure channel for sending emails. The port 25 is the default one for SMTP and it does not require authentication by default. There is another port who supports TLS and it is the port 465, but in this case it provides a secure way of sending emails by establishing an SSL/TLS encrypted connection from the very start of the communication and it is used for SMTPS (SMTP Secure). However, the port 465 was deprecated when STARTTLS became more popular, as STARTTLS allowed the standard Port 25 and Port 587 connections to be upgraded to encrypted connections without needing a dedicated port.
On the other hand, **Dovecot** is a high-performance mail delivery agent (MDA) with a focus on security. It manages email storage and provides access to emails for end users. Dovecot handles how users retrieve their emails from the server using the IMAP or POP3 protocols. In this case, the email server relies on port 993, which is the default port for IMAPS (IMAP Secure) and it is used when encryption is required from the start of the connection. Unlike Port 143, which is the default port for IMAP without encryption, when a client connects to Port 993, the communication is always encrypted using SSL/TLS. In the case of port 143, the communication between the client and the server is not encrypted by default, but to provide secure communication on Port 143, STARTTLS can be used.
In the case of the **server-config.sh** file, the only two additions that need to be made, compared to the file used as an example (Listing 5.2), are *service postfix start* and *service dovecot start* commands, which are used to start both postfix and dovecot.
Starting from the **Dockerfile** example (Listing 5.1), some additions have been made. The additions made are related to Postfix and Dovecot, starting with the installation of the necessary packages.

Listing 5.3: Dockerfile Email Server (Part 1)

```
RUN apt-get update && \
    apt-get install -y mailutils postfix postfix-pcre && \
    apt-get install -y dovecot-core dovecot-imapd dovecot-pop3d
```

The first thing did related to the setup for the email server is the creation of the user accounts, with usernames and passwords. Then a dynamic mechanism is employed to extract unique user and group identifiers (UIDs and GIDs) for each new user.These user credentials are then formatted for integration with Dovecot by appending each user's details, including a plain-text password and login identifiers, to the Dovecot user file. To ensure correct user handling, each user is associated with the respective UID and GID.

Listing 5.4: Dockerfile Email Server (Part 1)

```
# Create system users with specific UIDs and home directories
RUN adduser --gecos "" --home /home/sampointer sampointer && \
    echo "sampointer:sampointer" | chpasswd && \
    adduser --gecos "" --home /home/johndoe johndoe && \
    echo "johndoe:johndoe" | chpasswd

# Dynamically get the uid and gid for each user
RUN uid1=$(id -u sampointer) && gid1=$(id -g sampointer) && \
    uid2=$(id -u johndoe) && gid2=$(id -g johndoe) && \
    echo "sampointer:{PLAIN}sampointer:${uid1}:${gid1}::/home/sampointer:/bin/false" >>
        /etc/dovecot/users && \
    echo "johndoe:{PLAIN}johndoe:${uid2}:${gid2}::/home/johndoe:/bin/false" >>
        /etc/dovecot/users
```

A series of custom configuration files tailored for Dovecot and Postfix are copied into their respective directories within the container. These configurations define secure communication protocols, authentication methods, and logging settings, supporting a fully functional and secure mail server environment.

Listing 5.5: Dockerfile Email Server

```
COPY /10/10-ssl.conf /etc/dovecot/conf.d/10-ssl.conf
COPY /10/10-auth.conf /etc/dovecot/conf.d/10-auth.conf
COPY /10/10-logging.conf /etc/dovecot/conf.d/10-logging.conf
COPY /10/10-master.conf /etc/dovecot/conf.d/10-master.conf
COPY /20/20-imap.conf /etc/dovecot/conf.d/20-imap.conf
COPY /cf/auth-system.conf.ext /etc/dovecot/conf.d/auth-system.conf.ext
COPY /cf/main.cf /etc/postfix/main.cf
COPY /cf/master.cf /etc/postfix/master.cf
```

Postfix specific settings are applied to support domain-based email routing and virtual user mapping. A designated domain, *e-corp.com*, is assigned to the mail server, with virtual email addresses defined for each user. Then there is a command, *postmap*, that indexes these virtual email mappings into a database file, enabling streamlined and efficient email routing within the Postfix environment. SSL/TLS encryption is also configured by copying an SSL certificate and private key into the container. Both private key and certificate are generated using OpenSSL, with this command: *openssl req -new -x509 -days 365 -nodes -out postfix.pem -keyout postfix.key*.

Listing 5.6: Dockerfile Email Server

```
RUN sh -c 'echo "e-corp.com" >> /etc/mailname' && \
    sh -c 'echo "johndoe@e-corp.com johndoe" >> /etc/postfix/virtual' && \
    sh -c 'echo "sampointer@e-corp.com sampointer" >> /etc/postfix/virtual' && \
    postmap /etc/postfix/virtual

RUN chown root:root /etc/postfix/virtual /etc/postfix/virtual.db && \
    chmod 0600 /etc/postfix/virtual /etc/postfix/virtual.db

COPY cert/postfix.pem /etc/ssl/certs/postfix.pem
COPY cert/postfix.key /etc/ssl/private/postfix.key
```

5.3.1 Postfix Configuration

Postfix configuration is managed with files copied inside the container in the previously mentioned Dockerfile (Listing 5.5).
The **main.cf** file is the primary configuration file for Postfix and it is essential for defining how the Postfix server should handle various aspects of email processing, including security, domain recognition, relay permissions, and encryption. Postfix also identifies its operating domain and server hostname through the *mydomain* and *myhostname* settings. Here, mydomain defines the primary domain that emails appear to originate from, while myhostname sets the machine's unique identifier. Together, these settings help email clients recognize messages as legitimate, establishing consistency across email communications. The configuration includes settings that specify which domains the server will consider as local. In *mydestination*, Postfix identifies domains it will handle directly, meaning emails directed to any of these domains are treated as if they are local to this server. This setup allows the server to manage emails for specific domains, while *relayhost* remains empty, indicating that the server should send emails directly to recipient servers instead of routing through an intermediary.

Listing 5.7: main.cf Postfix (Part 1)

```
myhostname = e-corp.com
mydomain = e-corp.com
myorigin = $\dollar\$mydomain
mydestination = $\dollar$myhostname, mrrobot.com
relayhost =
```

There is also the possibility to impose network limitations via *mynetworks*, which limits the IP ranges permitted to relay mail through the server. Postfix is configured to accept connections on all available network interfaces through *inet_interfaces*, and both IPv4 and IPv6 protocols are enabled via *inet_protocols*, ensuring broad compatibility across diverse network environments.

Listing 5.8: main.cf Postfix (Part 2)

```
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
inet_interfaces = all
inet_protocols = all
```

Security is enhanced by Transport Layer Security (TLS) settings. The paths to the TLS certificate and key files are specified, enabling encrypted communications for email transfers, which reduces the risk of data interception during transmission. Additionally, Postfix's configuration specifies *smtpd_tls_security_level* and *smtp_tls_security_level* to *may*, allowing but not enforcing TLS on both incoming and outgoing connections. This provides compatibility with servers that may not support encryption, though a higher level of security could be enforced by requiring TLS.

Listing 5.9: main.cf Postfix (Part 3)

```
smtpd_tls_cert_file=/etc/ssl/certs/postfix.pem
smtpd_tls_key_file=/etc/ssl/private/postfix.key
smtp_tls_CApath=/etc/ssl/certs
smtp_tls_security_level=may
```

Postfix is also configured for virtual mailbox handling, enabling it to manage email storage for multiple domains. By specifying *virtual_mailbox_domains*, the server recognizes these domains as virtual and organizes messages accordingly in *virtual_mailbox_base*. The mappings for these mailboxes and any aliases are stored in hash maps located in */etc/postfix/virtual*, and these mappings direct emails to the appropriate storage locations.

Listing 5.10: main.cf Postfix (Part 4)

```
virtual_mailbox_domains = mrrobot.com
virtual_mailbox_base = /var/mail
virtual_mailbox_maps = hash:/etc/postfix/virtual
virtual_alias_maps = hash:/etc/postfix/virtual
```

The file also includes settings for Simple Authentication and Security Layer (SASL) authentication, which is crucial for secure user authentication. By specifying *smtpd_sasl_type* as Dovecot and setting the path to the Dovecot socket, the server allows authenticated users to send emails through the server, further preventing unauthorized use of the server for email relaying.

Listing 5.11: main.cf Postfix (Part 5)

```
smtpd_sasl_type = dovecot
smtpd_sasl_path = private/auth
```

In order to enable sending emails with STARTTLS, there is an addition to make to the **master.cf** file. The line *submission inet n - y - - smtpd* defines the service that listens for incoming email submissions on port 587. It specifies that the service is enabled (y) and uses the SMTP daemon (smtpd). The -o options following this line configure various security and authentication features, such as optional TLS encryption (if the client supports it), or enabling SASL authentication.

Listing 5.12: master.cf Postfix

```
submission inet n -y --smtpd
  -o syslog_name=postfix/submission
  -o smtpd_tls_security_level=may
  -o smtpd_sasl_auth_enable=yes
  -o smtpd_tls_auth_only=yes
  -o smtpd_recipient_restrictions=permit_sasl_authenticated,reject
  -o smtpd_relay_restrictions=permit_sasl_authenticated,reject
  -o milter_macro_daemon_name=ORIGINATING
```

To complete the configuration for Postfix, you need to configure dovecot file **10-master.conf** in order to make it handle email submissions, in a secure way, via port 587. By specifying *inet_listener submission port = 587* , Dovecot is set up to listen for incoming connections on this port, allowing clients to securely submit emails after authenticating.

Listing 5.13: 10-master.conf Postfix

```
service submission-login {
  inet_listener submission {
    port = 587
  }
}
```

5.3.2 Dovecot Configuration

Dovecot configuration is managed in the same way as the one for Postfix.
The first step to set up the Dovecot mail server, is to make it handle IMAP connections, specifying how the server should listen for incoming IMAP requests. This can be achieved by modifying the **10-master.conf** file. In this case Dovecot is configured to listen for IMAP connections on port 993. This port is reserved for IMAP over SSL/TLS, allowing clients to connect securely. The *ssl = yes* directive specifies that SSL/TLS encryption is required for any connections on this port, ensuring that all data transmitted between the client and server is encrypted and secure.

Listing 5.14: 10-master.conf Dovecot

```
service imap-login {
  inet_listener imaps {
    port = 993
    ssl = yes
  }
}
```

The next thing to do is check the configuration of the **10-ssl.conf** file, that should have ssl enabled and the paths to the certificate and the key of Dovecot.

Listing 5.15: 10-ssl.conf Dovecot

```
ssl = yes
ssl_cert = </etc/dovecot/private/dovecot.pem
ssl_key = </etc/dovecot/private/dovecot.key
```

It is also important to configure, from the **auth-system.conf.ext** file, how Dovecot manages user authentication and user information retrieval for email accounts. It specifies how email addresses are converted into usernames, along with the configuration for the password and user databases. The configuration begins with a directive that determines how Dovecot formats usernames for authentication purposes. By using the variable *%n*, Dovecot extracts the username portion of an email address, effectively removing the domain. This approach simplifies the authentication process, as it allows users to log in with just their usernames rather than requiring the full email

address. Following this, the *passdb* section is defined, which outlines the method Dovecot uses to verify user passwords. Here, the configuration specifies the use of a password file as the driver, with the path to this file indicated as */etc/dovecot/users*. This file contains user credentials, and Dovecot references it to authenticate users when they attempt to log in. Similarly, the *userdb* section is established to manage user-related information. It specifies that Dovecot will again utilize a password file as the driver, with the same path to */etc/dovecot/users*. This configuration allows Dovecot to look up essential user details, such as home directories, user IDs (UIDs), group IDs (GIDs), and other attributes necessary for properly handling user sessions and permissions.

Listing 5.16: auth-system.conf.ext Dovecot

```
auth_username_format = %n

passdb {
  driver = passwd-file
  args = /etc/dovecot/users
}

userdb {
  driver = passwd-file
  args = /etc/dovecot/users
}
```

The **auth-system.conf.ext** file must be included inside the **10-auth.conf**, with also the definition of which authentication mechanism is used. In this case authentication method used it PLAIN, which allows users to authenticate by sending their username and password in a straightforward manner.

Listing 5.17: 10-auth.conf Dovecot

```
auth_mechanisms = plain
!include auth-system.conf.ext
```

5.4 Database Server

The Database Server is placed in the dmz with the email server and it was created using PostgreSQL. This server is used to simulate a server of a company, containing, in this case, some data about projects and other work related things. During the simulations, the workers will interact with the database, adding and retrieving data from it.
As for the email server, starting from the **Dockerfile** proposed at the beginning of the chapter (Listing 5.1), there are few additions to make, starting from the necessary packages to install.

Listing 5.18: Dockerfile Database Server

```
1 RUN apt-get install -y postgresql postgresql-contrib
```

The next step is to add the commands to insert into the container one more bash script and two configuration files. The bash script is named *db_setup.sh* and it is needed to manage the creation of the database and everything related to it.

Listing 5.19: Dockerfile Database Server

```
1 COPY db_setup.sh /usr/local/bin/db_setup.sh
2 COPY conf/pg_hba.conf /etc/postgresql/16/main/pg_hba.conf
3 COPY conf/postgresql.conf /etc/postgresql/16/main/postgresql.conf
```

The **server-config.sh** file uses as a starting point the example shown at the start (Listing 5.2). In addition to the operations already present in the example file, there is a command to check whether the PostgreSQL data directory exists on the system, using the *$PGDATA* variable to locate it. If this directory is absent (indicating the database has not yet been initialized), it creates a new PostgreSQL data directory, invoking the *initdb* command under the postgres user to establish an initial environment for the database. This initialization step is essential on first run, ensuring PostgreSQL has a proper storage location. Following this, the script starts the PostgreSQL service to make the database accessible for subsequent configuration and use. Once the service is active, the script proceeds to configure PostgreSQL itself. It outputs a notification to signal the setup process and runs an external script, *db_setup.sh*, which contains instructions to establish a user, create a database, and configure permissions within PostgreSQL, providing the required setup for users and access control.

Listing 5.20: server-config.sh Database Server

```
1 PGDATA="/var/lib/postgresql/data"
2 if [ ! -d "$\dollar$PGDATA" ]; then
3     echo "Initializing PostgreSQL data directory..."
4     su -postgres -c "/usr/lib/postgresql/$\dollar$(pg_lsclusters -h | awk '{print
          $\dollar$1}')/bin/initdb -D $\dollar$PGDATA"
5 fi
6
7 service postgresql start
8 echo "Setting up PostgreSQL user, database, and permissions..."
9 source /usr/local/bin/db_setup.sh
```

The **db_setup.sh** script is designed to set up a PostgreSQL database with several user accounts, tables, and customized permissions for each user.
The script begins by creating a new database named *evil_corp*, with postgres as the owner.

Listing 5.21: db_setup.sh Database Server

```
1 su -postgres -c "psql -c \"CREATE DATABASE evil_corp;\""
```

It then creates multiple users assigning each a unique password. Each user is granted with connection privileges to *evil_corp*, allowing them to access the database, and usage privileges on the public schema,so they can see and use the tables within that schema.

Listing 5.22: db_setup.sh Database Server

```
su -postgres -c "psql -c \"CREATE USER johndoe WITH PASSWORD 'john';\""
su -postgres -c "psql -c \"GRANT CONNECT ON DATABASE evil_corp TO johndoe;\""
su -postgres -c "psql -d evil_corp -c \"GRANT USAGE ON SCHEMA public TO johndoe;\""

# Same for the other users
```

Subsequently, a set of tables is created within the *evil_corp* database. Each table has an id column (a unique identifier for each entry, generated automatically with a serial data type) and then additional columns to store specific data.

Listing 5.23: db_setup.sh Database Server

```
su -postgres -c "psql -d evil_corp -c \"CREATE TABLE project_info (id SERIAL PRIMARY KEY,
    section VARCHAR(100), detail VARCHAR(100), sub_detail TEXT);\""

su -postgres -c "psql -d evil_corp -c \"CREATE TABLE budget_info (id SERIAL PRIMARY KEY,
    section VARCHAR(100), detail VARCHAR(100), sub_detail TEXT);\""
```

The script then grants each user specific privileges on each table, tailored to their level of access. In this case there are SELECT, INSERT, UPDATE, DELETE, which allow, respectively, users to read, add, modify, or delete rows within the tables. Additionally, each user is granted permissions on the sequences associated with the id columns. This enables users to use the id column's automatic numbering feature for new entries.

Listing 5.24: db_setup.sh Database Server

```
su -postgres -c "psql -d evil_corp -c \"GRANT SELECT, INSERT, UPDATE, DELETE ON
    TABLE project_info TO johndoe;\""
su -postgres -c "psql -d evil_corp -c \"GRANT SELECT, INSERT, UPDATE, DELETE ON
    TABLE budget_info TO johndoe;\""

su -postgres -c "psql -d evil_corp -c \"GRANT USAGE, SELECT ON SEQUENCE
    project_info_id_seq TO johndoe;\""
su -postgres -c "psql -d evil_corp -c \"GRANT USAGE, SELECT ON SEQUENCE
    budget_info_id_seq TO johndoe;\""
```

5.4.1 PostgreSQL Configuration

The database server configuration is mainly managed through the two files copied into the Dockerfile. The **pg_hba.conf** file specifies which clients can connect, the methods used for authentication, and what access privileges are allowed. Each access rule includes several key fields: the *TYPE* field, indicating the connection type (such as local for Unix domain sockets or host for TCP/IP connections); the *DATABASE* field, specifying the applicable databases (e.g., all for all databases); the *USER* field, defining which PostgreSQL users the rule applies to (e.g., all for all users); the *ADDRESS* field, identifying the client's address (such as an IP address or range); and the *METHOD* field, specifying the authentication method (e.g., md5 for password-based authentication). This approach enables manual control over database access by specifying permitted IP address ranges.

Listing 5.25: pg_hba.conf PostgreSQL

```
# Allow access from 10.20.0.0/24 network
host all all 10.20.0.0/24 md5

# Allow access from 10.30.0.0/24 network
host all all 10.30.0.0/24 md5
```

The **postgresql.conf** file contains configuration settings for the PostgreSQL server. This file defines operational parameters such as networking, logging, and security settings. The *listen_addresses* parameter specifies which IP addresses PostgreSQL should listen to for incoming connections (in case there is '*', it means the server listens on all available IP addresses).

Listing 5.26: postgresql.conf PostgreSQL

```
listen_addresses = '*'
```

Then it is also possible to enable SSL, providing encryption for data sent between the client and server. In addition to enabling SSL, it is also possible to define the path to the SSL certificate file, which the server uses to verify its identity to clients, along with the path to the private key file for SSL.

Listing 5.27: postgresql.conf PostgreSQL (SSL)

```
ssl = on
ssl_cert_file = '/etc/ssl/certs/ssl-cert-snakeoil.pem'
ssl_key_file = '/etc/ssl/private/ssl-cert-snakeoil.key'
```

Additionally, the file configures logging settings to enhance monitoring and debugging. The server is set to log connection events, disconnection events, and query durations. This is achieved through the settings *log_connections = on*, *log_disconnections = on*, and *log_duration = on*. Furthermore, the server will log the hostname of the client connections with the directive *log_hostname = on*. The format of the log entries is specified by *log_line_prefix*, which is set to include the timestamp, process ID, user name, database name, and client host. Finally, the *log_statement* parameter is configured to log all SQL statements executed, contributing to comprehensive tracking of database activity. The timezone for logging is set to UTC, ensuring a consistent time reference across all logged events.

Listing 5.28: postgresql.conf PostgreSQL (logs)

```
log_connections = on
log_disconnections = on
log_duration = on
log_hostname = on
log_line_prefix = '%m [%p]: user=%u, db=%d, client=%h '
log_statement = 'all'
log_timezone = UTC
```

To view the logs, which provide insight into interactions with the database, the *cat /var/log/postgresql/postgresql-16-main.log* command can be used to display the content statically. Alternatively, the *tail -f /var/log/postgresql/postgresql-16-main.log* command allows for dynamic monitoring of log updates in real-time.

5.5 Firewall and Routers

The **firewall** is an important component in network security, because it functions as a barrier between trusted internal networks and untrusted external networks. Its primary function is to monitor and control incoming and outgoing network traffic based on predetermined security rules. By filtering traffic, firewalls protect sensitive data and systems from unauthorized access, attacks, and other security threats. In this configuration, the internal network is divided into three sub networks: LAN, WLAN, and guest. Additionally, there is a demilitarized zone (DMZ) where servers, such as email and database servers, are located, and an external network representing everything outside the company's network. This network segmentation enables management of access and communication permissions between the segments through firewall rules, enhancing security and control over data flow.
The **routers** play a fundamental role in the overall network configuration, as they facilitate communication between different networks, including the LAN, WLAN, guest network, and external connections, as well as the demilitarized zone (DMZ) where servers are hosted. By managing the flow of data, routers ensure that packets are directed to their intended destinations. When the destination device is within the same network as the sender, data can be transmitted directly. However, if the recipient is located in a different network, the data must first pass through the firewall before reaching its final destination. This routing process enables seamless communication between network components while maintaining security and control over data transmission.
The initial setup, for both firewall and routers, begins in the **Dockerfile** by establishing an Ubuntu-based Docker container and then involves installing core networking utilities. Essential tools, including *iptables* for traffic control and *iproute2* for advanced IP networking capabilities, are installed to enable precise management of network configurations and traffic. To ensure comprehensive network functionality, additional diagnostic and monitoring tools are added. Utilities such as *iputils-ping* for connectivity checks, *net-tools* for network configuration tasks, *traceroute* for tracing packet paths, *tshark* for network traffic capture, and *python3-pip* for Python package management are installed.

Listing 5.29: Dockerfile Firewall and Router

```
1 FROM ubuntu:latest
2
3 RUN apt-get update && \
4     apt-get install -y iptables iproute2 iputils-ping net-tools && \
5     apt-get install -y traceroute tshark sudo python3-pip && \
6     apt-get clean
```

Following the installation of necessary tools, the custom firewall (or router) configuration script, *firewall-config.sh* (or *router-config.sh*), is copied into the container. This script, which is subsequently made executable, contains the specific rules and logic to govern network traffic through the firewall or the router. An additional script, *capture_traffic.py*, is included and also used by various containers to capture network traffic using Wireshark. The capture file generated is stored in a dedicated folder created specifically for this purpose, allowing for organized traffic monitoring and analysis.

Listing 5.30: Dockerfile Firewall (Part 2)

```
1 COPY firewall-config.sh /usr/local/bin/firewall-config.sh
2 RUN chmod +x /usr/local/bin/firewall-config.sh
3
4 COPY pythonScript/capture_traffic.py /usr/local/bin/capture_traffic.py
5 RUN mkdir -p /home/captureTraffic
```

The primary firewall configuration is managed within the **firewall-config.sh** script and it is similar to the one used for the router, which includes custom routing and then, only for the firewall, filtering rules to control network traffic effectively. It begins by enabling IP forwarding, which allows the container to route packets between different networks. This is a crucial step in configuring the firewall to act as a router that controls traffic flow. Then specific routes are established to direct traffic between the firewall and several internal networks: the local area

network (LAN), wireless LAN (WLAN) and guest network. For example, traffic from the LAN is routed through 192.168.2.2, while guest traffic uses 192.168.4.2.

Listing 5.31: Firewall Example

```
sysctl -w net.ipv4.ip_forward=1

ip route add 10.20.0.0/24 via 192.168.2.2 # firewall -> LAN
ip route add 10.30.0.0/24 via 192.168.3.2 # firewall -> WLAN
ip route add 10.40.0.0/24 via 192.168.4.2 # firewall -> guest
```

To maintain Network Address Translation (NAT) after changing the gateway, the script identifies the network interfaces associated with two external IPs (198.51.100.1 and 198.51.101.1). Using these interface names, it configures NAT on both interfaces to allow outgoing packets from internal networks to appear as if they originated from the firewall's external IP addresses. This is achieved using the iptables MASQUERADE rule, which is essential for concealing internal IPs. In this configuration, two separate network interfaces are assigned to separate external IPs. One interface is connected to the external network, while the other is designated for the specific attacker's location in the simulation environment.

Listing 5.32: Firewall Example

```
OUTSIDE_NET_IFACE=$(sh -c "ip -o -4 addr show | grep '198.51.100.1' | cut -d' ' -f2")
iptables -t nat -A POSTROUTING -o "$OUTSIDE_NET_IFACE" -j MASQUERADE

EXT_NET_IFACE=$(sh -c "ip -o -4 addr show | grep '198.51.101.1' | cut -d' ' -f2")
iptables -t nat -A POSTROUTING -o "$EXT_NET_IFACE" -j MASQUERADE
```

The firewall rules define which types of traffic are permitted or blocked across different network segments. For instance, traffic originating from the guest network (10.40.0.0/24) is blocked when attempting to access the LAN (10.20.0.0/24) or WLAN (10.30.0.0/24), unless it is part of an established connection. Return traffic for already established connections is generally allowed to ensure that ongoing sessions can continue uninterrupted. Several rules grant controlled access from internal networks to external destinations, allowing the firewall to distinguish between different internal sources. For example, DMZ (192.168.1.0/24) traffic is permitted to communicate with the external network; however, external access to internal networks (10.20.0.0/24, 10.30.0.0/24, and 10.40.0.0/24) is strictly denied, adding a protective barrier.

Listing 5.33: Firewall Example

```
iptables -A FORWARD -s 10.40.0.0/24 -d 10.20.0.0/24 -m state ! --state
    ESTABLISHED,RELATED -j DROP
iptables -A FORWARD -s 10.40.0.0/24 -d 10.30.0.0/24 -m state ! --state
    ESTABLISHED,RELATED -j DROP

iptables -A FORWARD -i $OUTSIDE_NET_IFACE -o dmz_net -d 192.168.1.0/24 -j ACCEPT

iptables -A FORWARD -i $OUTSIDE_NET_IFACE -o inside1_net -d 10.20.0.0/24 -j DROP
iptables -A FORWARD -i $OUTSIDE_NET_IFACE -o inside2_net -d 10.30.0.0/24 -j DROP
iptables -A FORWARD -i $OUTSIDE_NET_IFACE -o inside3_net -d 10.40.0.0/24 -j DROP
```

Additional rules ensure that internal networks (inside1, inside2, and inside3) can reach the outside network, establishing that only permitted internal traffic can initiate external connections. Furthermore, the script configures permissions for the DMZ to access specific internal networks, allowing limited, monitored interaction with internal segments, such as inside1 (10.20.0.0/24), inside2 (10.30.0.0/24), and inside3 (10.40.0.0/24).

Listing 5.34: Firewall Example

```
iptables -A FORWARD -i inside1_net -o $OUTSIDE_NET_IFACE -j ACCEPT

iptables -A FORWARD -s 192.168.1.0/24 -d 10.20.0.0/24 -j ACCEPT
iptables -A FORWARD -s 192.168.1.0/24 -d 10.30.0.0/24 -j ACCEPT
iptables -A FORWARD -s 192.168.1.0/24 -d 10.40.0.0/24 -j ACCEPT
```

The default iptables policy is set to drop all incoming traffic to the FORWARD chain by default, enforcing a "deny-all" approach unless traffic is explicitly allowed by the rules outlined above. This configuration ensures that any unspecified or unexpected traffic is automatically blocked, adding an additional layer of security.

Listing 5.35: Firewall Example

```
1 iptables -P FORWARD DROP
```

5.6 Workers' Computer

Each container representing a worker's computer within the simulated company environment plays an essential role, as each can be customized to vary in content and usage. This flexibility allows for diverse configurations, enabling realistic simulation of different roles and activities within the organization. The configuration is managed using a Dockerfile and a bash script, while certain behaviors, such as internet browsing, are simulated through Python scripts.
The **Dockerfile** specifies Ubuntu version 22.04 instead of the latest release due to compatibility issues with certain audio and graphics libraries, such as *libasound2* and *libgl1-mesa-glx*, which are not available in the most recent Ubuntu versions. To simplify setup and avoid interruptions, the front end is set to non-interactive mode, which prevents configuration prompts from appearing while packages are being installed.

Listing 5.36: Dockerfile Computer

```
1 FROM ubuntu:22.04
2 ENV DEBIAN_FRONTEND=noninteractive
```

To prepare the container, it installs a suite of essential network utilities like *ping*, *traceroute*, *dnsutils*, and additional utilities such as *mutt* to support email handling and *postgresql-client* to interact with the database. Additional libraries, such as *libgtk-3-0*, *libdbus-glib-1-2* and *libx11-xcb1*, are installed to support applications that require a graphical interface, such as web browsers, even in automated or headless testing setups. These libraries enable smooth visual rendering, effective management of window and UI elements, and efficient handling of graphics processing. This configuration ensures that applications like Firefox have all the necessary visual and functional components for reliable browser-based testing.

Listing 5.37: Dockerfile Computer (Part 2)

```
1 RUN apt-get update && \
2     apt-get install -y libgtk-3-0 libdbus-glib-1-2 libx11-xcb1 libxt6 && \
3     apt-get install -y libxrender1 libxcomposite1 libpangocairo-1.0-0 libvulkan1 && \
4     apt-get install -y libasound2 libgl1-mesa-glx
```

In this part of the setup, *geckodriver* is downloaded and installed, along with *Firefox* and the *Selenium* framework. Geckodriver is essential because it acts as a bridge between Selenium and Firefox, enabling Selenium to control and automate Firefox for web browsing.

Listing 5.38: Dockerfile Computer (Part 3)

```
1  RUN wget
       https://github.com/mozilla/geckodriver/releases/download/v0.35.0/geckodriver-version_os.tar.gz
2  RUN tar -xzvf geckodriver-v0.35.0-linux64.tar.gz
3  RUN rm geckodriver-v0.35.0-linux64.tar.gz
4  RUN mv geckodriver /usr/local/bin/
5
6  RUN wget -O /tmp/firefox.tar.bz2
       "https://download.mozilla.org/?product=firefox-latest&os=linux64&lang=en-US"
7  RUN tar -xjf /tmp/firefox.tar.bz2 -C /opt/
8  RUN ln -s /opt/firefox/firefox /usr/local/bin/
9  RUN rm /tmp/firefox.tar.bz2
10
11 RUN pip3 install selenium
```

This is followed by copying several scripts, including a bash script named *fake_history* used to fill the file *.bash_history* with some commands. Additional Python scripts are also copied to the */usr/local/bin/* directory, each of which appears to automate various activities, like network traffic and web browsing. A system user is then created with specific configurations to enhance permissions by adding the user to the sudoers group without requiring a password.

Listing 5.39: Dockerfile Computer (Part 4)

```
RUN adduser --gecos "" --home /home/sarahwilliams sarahwilliams && \
    echo "sarahwilliams:sarahwilliams" | chpasswd && \
    usermod -aG sudo sarahwilliams

RUN echo "sarahwilliams ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers
```

A custom configuration file, essential for managing interactions with the email server (named *muttrc*), is copied into the user's home directory, with permissions adjusted to ensure correct ownership and functionality. Empty files named *sent* and *inbox* are created to store sent and received emails, respectively, simulating typical user email behavior and structures. The script then configures a Firefox user profile in headless mode, ensuring that it can run automated browser actions tied to this user. Finally, several directories are created within the user's home directory to simulate standard storage areas.

Listing 5.40: Dockerfile Computer (Part 5)

```
RUN mkdir -p /home/sarahwilliams/.mozilla/firefox/profiles
RUN firefox --headless -CreateProfile "SarahWilliams
    /home/sarahwilliams/.mozilla/firefox/profiles/sarahwilliams"
```

An environment variable, *SSLKEYLOGFILE*, is set in the user's *.bashrc* file, specifying the path to a file where SSL/TLS session keys will be logged. This setup allows for the capture and recording of cryptographic keys used during secure network communications, enabling detailed traffic analysis. Additionally, the script copies a file containing SSL certificates to the user's home directory. The *.mutt_certificates* file stores trusted certificates, enabling secure connections without triggering certificate warnings. This setup eliminates the need for manually accepting certificates when using Mutt to send or retrieve emails.
The **setup.sh** file is a script needed to complete the setup of the container. This script configures the network settings and system environment for the user within the container. Initially, it updates the DNS resolver configuration to use Google's public DNS servers, specifically 8.8.8.8 and 8.8.4.4, ensuring reliable domain name resolution. Then the script removes the default gateway pointing to 10.20.0.254 and sets a new default gateway to 10.20.0.253, which is the router used in the LAN network. For hostname resolution, the script updates the /etc/hosts file by associating IP addresses with domain names for internal servers.

Listing 5.41: setup.sh Computer Example

```
echo "192.168.1.4 email_server" >> /etc/hosts
echo "192.168.1.2 database_server" >> /etc/hosts
```

The script also generates several files to populate the container with data, enhancing the realism of the environment. It then modifies the file permissions for specific directories and files to ensure the created user has the appropriate access rights. Furthermore, the script sources the *fake_history.sh* file, which adds a series of shell commands to the *.bash_history* file.

Listing 5.42: fake_history.sh Computer Example

```
echo 'cd new_ideas' >> /home/sarahwilliams/.bash_history && history -a
echo 'ls -la' >> /home/sarahwilliams/.bash_history && history -a
echo 'cd /' >> /home/sarahwilliams/.bash_history && history -a
echo 'clear' >> /home/sarahwilliams/.bash_history && history -a
```

Listing 5.43: setup.sh Computer Example

```
chown sarahwilliams:sarahwilliams /home/sarahwilliams/sent
chown sarahwilliams:sarahwilliams /home/sarahwilliams/inbox
chown -R sarahwilliams:sarahwilliams /home/sarahwilliams

source ./usr/local/bin/fake_history.sh
```

5.6.1 Interaction With Database Server

To enable database management and access on a computer in a networked environment, a PostgreSQL client must first be installed. In this case *apt-get install -y postgresql-client* is used on the computer, enabling it to communicate with the PostgreSQL database server. Once the computer is configured, a command can be issued to access the database. To do so, it is necessary to specify the hostname (*-h database_server*) so that the client knows the server's location. An example command for accessing the database is:

Listing 5.44: Computer Database Connection

```
1 PGPASSWORD='sarah' psql -h database_server -d evil_corp -U sarahwilliams
```

This command connects to the *evil_corp* database on *database_server* using the username *sarahwilliams*, with *sarah* as the password. By running this command, a user gains access to the database and can perform various operations.
For example, to retrieve data from a table, the following command can be used:

Listing 5.45: Computer Database Retrieve Data

```
1 PGPASSWORD='sarah' psql -h database_server -d evil_corp -U sarahwilliams -c "SELECT *
      FROM project_info;"
```

This command executes a SQL query to retrieve all data from the project_info table.
Additionally, it is possible to insert data directly into the table or to import data from an external file directly into a table with the `\COPY` command, as shown here:

Listing 5.46: Computer Database Insert Data From File

```
1 PGPASSWORD='sarah' psql -d evil_corp -U sarahwilliams -c "INSERT INTO project_info (section,
      detail, sub_detail) VALUES ('section', 'detail', 'sub_detail');"
2 PGPASSWORD='sarah' psql -h database_server -d evil_corp -U sarahwilliams -c "\COPY
      project_info(section, detail, sub_detail) FROM './path/projectOverview.csv' DELIMITER ','
      CSV HEADER;"
```

5.6.2 Interaction With Email Server

To ensure proper interaction between the container and the email server using mutt, it is essential to correctly configure the **.muttrc** file. The .muttrc file typically contains settings for both incoming and outgoing email, including server configurations (IMAP and SMTP), authentication details, and user information such as the email address and display name. By specifying parameters like the IMAP server's address and port number, the file enables secure email retrieval, while the SMTP settings ensure that outgoing emails are sent securely.
The IMAP configuration specifies the user's email address and password for accessing the email server. The file sets the IMAP server to connect securely over port 993 using the *email_server* hostname, which is expected to support encrypted communication. It defines the inbox folder to be accessed as the user's default *INBOX* and specifies cache directories to store headers and message bodies for efficient mail retrieval.

Listing 5.47: muttrc Computer Example (IMAP)

```
1 set imap_user = "sarahwilliams@e-corp.com"
2 set imap_pass = "sarahwilliams"
3 set folder = "imaps://email_server:993"
4 set spoolfile = "+INBOX"
5 set header_cache = "~/.mutt/cache/headers"
6 set message_cachedir = "~/.mutt/cache/bodies"
```

For outgoing mail, the configuration provides the necessary settings for connecting to the SMTP server. The *smtp_url* is defined, using the user's credentials and the server's address for sending emails via port 587 with STARTTLS encryption. Additionally, the user's email address and real name are set for identifying the sender.

Listing 5.48: muttrc Computer Example (SMTP)

```
set smtp_url = "smtp://sarahwilliams:sarahwilliams@email_server:587" # STARTTLS
set smtp_pass = "sarahwilliams"
set from = "sarahwilliams@e-corp.com"
set realname = "Sarah Williams"
```

The configuration also ensures secure communication by enforcing the use of TLS for all email transmissions. the settings enable the inclusion of the original message in replies, providing context to recipients when responding. The configuration also allows for the automatic setting of a "Reply-To" address, which can be directed to the sender of the email. These adjustments make it easier to automate email communication, ensuring a more smooth and efficient interaction.

Listing 5.49: muttrc Computer Example

```
set ssl_force_tls = yes
set ssl_starttls = yes

set include = yes
set reply_to = yes
```

Several methods are available for sending a new email. The first method involves launching Mutt from the command line, which then displays the list of received emails and allows various actions to be performed. This approach is particularly useful when working with self-signed certificates, as it enables manual acceptance or rejection of certificates both when Mutt is opened and when an email is sent. By pressing m, a new email can be composed; r allows a reply to the currently selected email; d deletes the selected email; and q exits Mutt.
Another option is to send an email directly from the command line with a single command, though issues may arise if the required certificates have not been previously accepted.

Listing 5.50: Command Line Mutt Example

```
echo "Email body text" | mutt -s "Subject" receiver@e-corp.com
echo "Email body text" | mutt -s "Subject" -a /path/to/attachment --receiver@e-corp.com
```

5.7 Automation Scripts

Scripts are essential to these simulations, as they enable the reproduction of various behaviors and streamline the process of replicating specific scenarios, even with modifications. In this context, scripts have been employed to automate internet searches, capture network traffic with Wireshark in different containers, and simulate the actions of different employees. This includes generating different email conversations with modified send dates, managed through the use of *faketime* and the *date –set* command, as well as performing database operations with similarly modified timestamps. *Faketime* is used to modify the system time for specific processes without affecting the actual system clock (like when using tools such as Mutt or PostgreSQL client from the command line). The command *date –set* is used to change the system's date and time, allowing a specific date and time to be set on the system clock. In this context, it is employed, for example, with the email server to alter the timestamps of sent and received emails

5.7.1 Wireshark Capture (capture_traffic.py)

This script is designed to capture network traffic on a computer using the tshark tool, which is the command-line counterpart to Wireshark, a network protocol analyzer. It operates by starting a capture process, saving the data to a specified file, and then stopping the capture after a user input.
At the beginning, the script sets up the environment by importing the necessary libraries: time to handle delays and subprocess to manage external processes. It defines two main functions: *start_capture* and *stop_capture*. The **start_capture** function is responsible for initiating the packet capture using tshark. It runs the tshark command with specific arguments: it saves the captured traffic to a file, limits the file size to a specified value, and executes the command in the background. It uses *subprocess.Popen* to run the command without blocking the rest of the script, allowing it to continue its operations. The **stop_capture** function is used to stop the tshark process. It does so by calling terminate() on the running tshark process, which signals the process to stop, and then waits for the process to finish using wait().
In the **main** the user is notified that the capture process is starting. The script waits for some seconds using *time.sleep()* and then ask the user to press Enter whenever he wants to stop the capture.

5.7.2 Web Navigation with Selenium (start_web_nav.py)

This script automates web browsing, network traffic capture, and file downloading using Python, while manipulating the system time to simulate actions at different timestamps. It utilizes several tools, such as Selenium for browser automation, tshark for network packet capturing, and system commands to alter the time.
The part about using tshark is the same as the *capture_traffic.py* script. The main focus in this case is on web browsing done with selenium. It begins by initializing a browser session using Selenium and configuring it to run without a graphical interface (headless mode).

Listing 5.51: Firefox Setup

```
profile_path = "/home/sarahwilliams/.mozilla/firefox/profiles/sarahwilliams"
download_dir = "/home/sarahwilliams/downloads"

options = Options()
options.add_argument("--headless")
options.add_argument("-profile")
options.add_argument(profile_path)
# Setting preferences for downloading files
options.set_preference("browser.download.folderList", 2)
options.set_preference("browser.download.dir", download_dir)
options.set_preference("browser.download.manager.showWhenStarting", False)
options.set_preference("browser.helperApps.neverAsk.saveToDisk", "application/pdf")
options.set_preference("pdfjs.disabled", True)
```

```
options.set_preference("acceptInsecureCerts", True) # Needed for fake date
options.set_preference("security.cert_pinning.enforcement_level", 0)
options.set_preference("security.tls.version.min", 1)
options.set_preference("security.tls.version.max", 3)
```

Then, it proceeds to manipulate the system time using the **set_fake_time** function before navigating to specified URLs, allowing actions such as accepting cookies, searching, or downloading files, all while the system time is faked to simulate specific times.

Listing 5.52: Navigation With Selenium

```
url = "https://killedbygoogle.com"
website_name = "Killed By Google"

date_obj = datetime.strptime(initial_fake_date, "%Y-%m-%d %H:%M:%S")
fake_date = str(date_obj + timedelta(days=1, hours=1, minutes=33, seconds=43))

# Set the fake date on the device and perform the search
navigate_web_page(options, url, website_name, fake_date)
```

The function **navigate_web_page** is used to browse the internet by visiting a specified URL. In contrast, the **search_with_google** function performs a search on Google by entering a set of keywords, then selecting and clicking on the first result, while also handling cookie acceptance if necessary. The last function, **download_pdf_function**, allows for downloading a PDF by accessing a given URL.

5.7.3 Email and Database Automation

This setup involves a collection of Python scripts that work together to simulate email conversations among the company's workers and their interactions with the database. These actions are coordinated using both the *faketime* and *date –set* commands, allowing for the manual selection of dates and times to create realistic scenarios.
The first script, **basic_email_setup.py**, is designed to sequentially execute Python scripts simulating activities for individual days by using subprocess.run. A second version of this file, referred to as **days_additional.py**, serves the same general purpose. However, in *days_additional.py*, the content of the daily simulation scripts can vary depending on the specific type of attack being recreated.

Listing 5.53: basic_email_setup.py Script Example

```
import subprocess

subprocess.run(["python3", "./days/day01.py"])
subprocess.run(["python3", "./days/day02.py"])
```

Each **dayXX.py** script simulates a sequence of email exchanges between colleagues and interactions with the database in a time span of one day. The simulation starts from a shared initial fake date, which is adjusted to reflect the specific date and time of the current simulated day. For example, in the script representing day 3, two days are added to the initial starting date to set the correct time-frame.

Listing 5.54: day03.py Script Example

```
import time
from write_emails import *
from datetime import datetime, timedelta
from database_actions.faketime_operations import *

date_obj = datetime.strptime(initial_fake_date, "%Y-%m-%d %H:%M:%S")
fake_date = str(date_obj + timedelta(days=2, minutes=4, seconds=16))
```

The imports include two Python scripts, *write_emails* and *faketime_operations*, which contain functions specifically designed to manage, respectively, email communications and database interactions.
An example of this script's function is the process of sending an email from one user to another. Initially, the subject and body of the email are defined, followed by specifying the container where the command will be executed, the associated username, and the email address of the recipient. This setup is followed by the creation of the new fake date and then by the calls to two functions: the first function, *set_fake_time*, adjusts the date on the email server, and the second function, *send_email*, sends the email.

Listing 5.55: day03.py Send Email Example

```
email_subject = "Email Subject"
email_body = "Email body"
container = johndoe_computer2
receiver = sarahwilliams_email
sender = "johndoe"

date_obj = datetime.strptime(fake_date, "%Y-%m-%d %H:%M:%S")
fake_date = str(date_obj + timedelta(hours=1, minutes=11, seconds=20))

set_fake_time(email_server_container, fake_date)
send_email(container, fake_date, receiver, email_subject, email_body, sender)
```

Automating email responses is also possible, following a process similar to sending a standard email. However, the procedure begins with calling a function, *save_email*, which saves emails locally. The subject for the response must match that of the original email, with "Re: " added at the beginning. Specifying the recipient is unnecessary, as the *send_email_reply* function retrieves the original sender's address from the email being replied to.

Listing 5.56: day03.py Reply Email Example

```
email_subject = "Re: Monthly Report Submission"
email_body = "Response Email body"
container = oliviamurphy_pc4
sender = "oliviamurphy"

date_obj = datetime.strptime(fake_date, "%Y-%m-%d %H:%M:%S")
fake_date = str(date_obj + timedelta(minutes=45, seconds=55))

set_fake_time(email_server_container, fake_date)
send_email_reply(container, fake_date, email_body, sender, email_subject)
```

For database interactions, an example involves a user extracting data from the database. First, the necessary parameters are defined, including the container in which the command is executed, database credentials, the target table for data extraction, the database name, and the username associated with the container. Once these parameters are set, the appropriate functions are called to adjust the date on the database server and the one to interact with the database.

Listing 5.57: day03.py Database Interaction Example

```
container = sampointer_pc1
db_username = "sampointer"
db_password = "sam"
table_name = "team_involved"
db_name = "evil_corp"
user = "sampointer"

date_obj = datetime.strptime(fake_date, "%Y-%m-%d %H:%M:%S")
fake_date = str(date_obj + timedelta(minutes=5, seconds=31))

set_fake_time(database_server_container, fake_date)
retrieve_data(container, fake_date, db_username, db_password, table_name, db_name, user)
```

The **write_emails.py** script initially defines all the variables, including the containers, the email addresses of the various users, and the fake start date.

Listing 5.58: write_emails.py Variable Example

```
jamesfoster_computer4 = "computer4"
jamesfoster_email = "jamesfoster@e-corp.com"
initial_fake_date = "2024-10-21 09:03:23"
```

The first function is *set_fake_time* and its job is to change the date and time of a container using *date –set*. This allows the script to control the progression of time, ensuring that email communications and server operations appear to take place at the desired points in the simulated timeline. Email sending is managed through the *send_email* function, which constructs an email with a subject and body, and then dispatches it using a specified user account and the manipulated system time. The emails are sent using *mutt* with *faketime* to manipulate the time.
Similarly, the *send_email_reply* function enables replying to emails by extracting relevant email data, such as the original sender and message ID, from an incoming email. This is done by searching for the original message in the user's inbox, extracting its details, and constructing a reply that includes a quoted version of the previous message.
The script also handles the saving of emails using the *save_email* function, which attempts to run a Python script designed for email saving on a designated container. If the operation fails due to a timeout, the script retries the process after restarting key email services (Dovecot and Postfix) on the email server, ensuring that the email-saving operation is completed. These problems related to saving emails locally can be due to date and time manipulation in the email server, for this reason, before trying again, both Dovecot and Postfix are restarted.
The **faketime_operations.py** job is to provide the functions needed to interact with the database. One of the main functions is *query_fakedate*, which performs a database query on a specified table within a container. It first checks if a directory exists for storing the query results. If the directory is absent, the script creates it. The query is then executed using PostgreSQL, and the results are saved as a CSV file in the designated directory, with the operation occurring under the simulated date provided with faketime. This is the function used in the Data Breach attack simulation with the *merge_files* one, which handles the merging of multiple CSV files. The script merges the files from a target folder into a single output file, appending the contents of each CSV file sequentially, while also adding the filename as a header for clarity.
The *insert_data* function is designed to insert new data into a specified table in the database. It constructs an SQL INSERT query based on the provided data and executes it using the psql command within the container. The insertion occurs under the specified fake date, allowing for time-sensitive operations to be logged correctly.
Lastly, the *retrieve_data* function retrieves all data from a specific table in the database. Similar to the insertion function, it constructs an SQL SELECT query and runs it inside the container under the simulated date, ensuring that the data retrieval occurs in sync with the controlled timeline.

Chapter 6

Data Breach Simulation

6.1 Description of the Architecture

The architecture used in this simulated attack scenario involves an external attacker, Elliot Alderson, and an accomplice within the company, Sarah Williams. Sarah interacts with other employees via email and engages with the company database, where she adds or retrieves information as part of her role. Among these employees, Sam Pointer becomes an unknowing victim of the breach when Sarah, under a pretense, requests to use his credentials to access the database. Once granted access, she extracts sensitive data, saves it in multiples files, and then merges these files into a single document.
To avoid detection, Sarah names the folders in a way that implies the files are personal rather than work-related. The merged file, containing all extracted data, is labeled "doctorApp.csv" and is eventually attached to an email that Sarah sends to Elliot. After sending this email, she deletes the folder containing the merged file to hide her actions. Throughout the preparation and execution of the attack, Elliot and Sarah exchange a series of emails to coordinate their efforts. Alongside these messages, Sarah continues routine email communications with other employees to maintain the appearance of normal operations.
The company structure was created based on the basics covered in the chapter on docker setup (chapter 5), with the external attacker container as the only addition. The data breach simulation is entirely handled by the previously mentioned python scripts (subsection 5.7.3). In each simulated day, emails are sent and database operations are performed. In the middle of these daily activities, there are suspicious emails between Sarah and Elliot in which they organize how to proceed to complete their work.

6.2 How to Detect a Data Breach

Data breaches can be detected through a variety of methods that focus on identifying and responding to suspicious activities across systems. In this context, **User Behavior Analytics** (UBA) plays a critical role by establishing a baseline of typical user activities through the monitoring of login patterns, file access behavior, and application usage. Any deviations from these established norms, such as logins from unexpected locations or devices, access to unusual datasets, or the retrieval of large amounts of information, can serve as indicators of potential unauthorized access or malicious activity. Complementing this, there is **Privileged Access Management** (PAM), which is another strategy that focuses on controlling and monitoring high-level accounts. PAM limits access to sensitive data, allowing it only for users with a clear, verified need. Monitoring privileged accounts closely helps detect suspicious behavior, as these accounts are often targeted in attacks. Attempts to access restricted areas or escalate privileges can trigger alerts for further investigation. There is also **Database Activity Monitoring** (DAM), which tracks and analyzes database interactions for unusual query patterns or unauthorized requests, flagging privilege escalations and other anomalies that could compromise sensitive information. Similarly, network and

email activities are essential areas of focus. **Email monitoring** and **Data Loss Prevention** (DLP) systems scan outgoing communications for sensitive content, preventing data exfiltration through external emails, particularly when unusual attachments or transmission times are detected. On a larger scale, **network monitoring** identifies anomalous traffic patterns or data transfers that may signal exfiltration attempts. Tools like *Network Intrusion Detection Systems* (NIDS) enhance this by identifying known patterns of malicious activity, while monitoring email gateways ensures that sensitive data is not transmitted to untrusted or external domains. In the end, there are **Security Information and Event Management** (SIEM) systems that also play a crucial role in identifying potential data breaches. By aggregating logs from various systems, such as databases, endpoint devices, and email servers, SIEM solutions provide centralized logging that enables detection of suspicious patterns across multiple sources. This centralized approach allows for correlation of events; for example, if a user logs in from an unusual IP address, runs a large database query, and subsequently sends an external email, these combined activities may elevate the incident to a high-priority alert. In response to detected anomalies, SIEM systems can automatically initiate incident response actions, such as blocking user access, notifying the security team, or isolating potentially compromised endpoints, thereby helping to contain and mitigate security threats.

6.3 How to Investigate a Data Breach

Once receiving notification of a potential data breach, it is essential to determine which systems or data have been impacted and how. For example, an anomaly could be detected if a user accesses the database from a device with a different IP address than usual. This behavior triggers monitoring tools that evaluate user activity patterns to flag actions that deviate from the norm, thereby marking them as potentially suspicious.
Initially, the database is analyzed to examine the logs thoroughly, allowing for a clearer identification of the users and data involved in the incident. The logs reveal that the user *sampointer*, typically accesses the database from a computer with the IP address *10.30.0.1*. However, there are interactions originating from a different IP address, *10.20.0.1*, which appear irregular. These suspicious activities occurred over a two-day period.

Listing 6.1: First Suspected Database Interaction

```
1 2024-10-26 15:27:47.558 UTC [750]: user=[unknown], db=[unknown], client=10.20.0.1 LOG:
      connection received: host=10.20.0.1 port=59898
2 2024-10-26 15:27:47.571 UTC [750]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      connection authenticated: identity="sampointer" method=md5
      (/etc/postgresql/16/main/pg_hba.conf:25)
3 2024-10-26 15:27:47.571 UTC [750]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      connection authorized: user=sampointer database=evil_corp application_name=psql SSL
      enabled (protocol=TLSv1.3, cipher=TLS_AES_256_GCM_SHA384, bits=256)
4 2024-10-26 15:27:47.572 UTC [750]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      statement: SELECT * FROM project_info;
5 2024-10-26 15:27:47.573 UTC [750]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      duration: 0.528 ms
6 2024-10-26 15:27:47.573 UTC [750]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      disconnection: session time: 0:00:00.051 user=sampointer database=evil_corp host=10.20.0.1
      port=59898
```

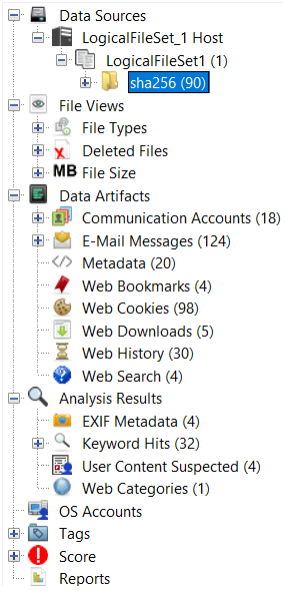
This log represents a sequence of events occurring within a database connection session. It begins with the database server receiving a connection request from a client located at IP address *10.20.0.1*. At the beginning, the log identifies neither a user nor a database, which is typical until authentication is complete. Shortly after, the user, identified as *sampointer*, successfully authenticates to the database *evil_corp* using MD5 hashing. Following this authentication, the database authorizes the connection, confirming access rights for user *sampointer* to the specified database. After the successful authorization, the user *sampointer* issues an SQL statement, specifically a *SELECT * FROM project_info* command. The session concludes almost immediately after the query, with a disconnection log entry indicating the overall session duration.

Listing 6.2: Last Suspected Database Interaction

```
1 2024-10-27 10:26:49.533 UTC [814]: user=[unknown], db=[unknown], client=10.20.0.1 LOG:
      connection received: host=10.20.0.1 port=45096
2 2024-10-27 10:26:49.546 UTC [814]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      connection authenticated: identity="sampointer" method=md5
      (/etc/postgresql/16/main/pg_hba.conf:25)
3 2024-10-27 10:26:49.546 UTC [814]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      connection authorized: user=sampointer database=evil_corp application_name=psql SSL
      enabled (protocol=TLSv1.3, cipher=TLS_AES_256_GCM_SHA384, bits=256)
4 2024-10-27 10:26:49.547 UTC [814]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      statement: SELECT * FROM team_involved;
5 2024-10-27 10:26:49.548 UTC [814]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      duration: 0.443 ms
6 2024-10-27 10:26:49.548 UTC [814]: user=sampointer, db=evil_corp, client=10.20.0.1 LOG:
      disconnection: session time: 0:00:00.036 user=sampointer database=evil_corp host=10.20.0.1
      port=45096
```

After understanding that the device with the IP address *10.20.0.1* has conducted operations using an unusual account, further analysis of the device's image can be done using forensic tools like Autopsy. In the context of this simulation, the image is obtained through the specified Docker commands (commit and save, subsection 5.2.1) and subsequently loaded into Autopsy. Initially, the *Embedded File Extraction* module is executed, and after its successful completion, additional modules, such as *Recent Activity*, *Email Parser*, and *File Type Identification*, can be applied to extract as much information as possible from the image.

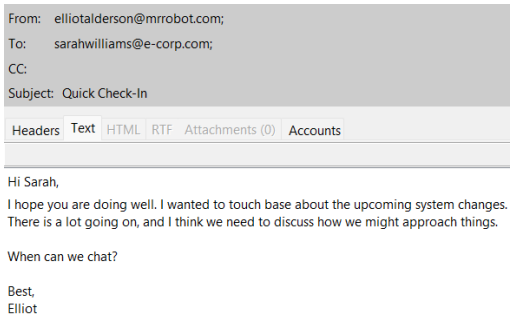
Figure 6.1: Result Run Ingest Modules in Autopsy



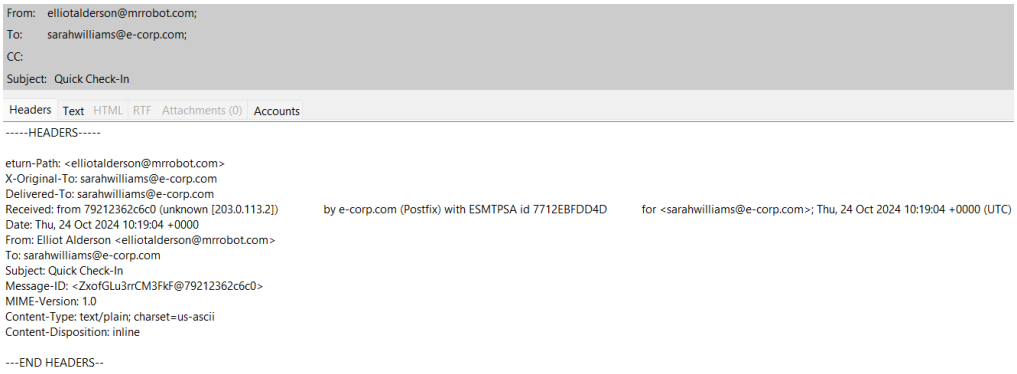
The output shown in Figure 6.1 displays the results from the ingest modules executed on the container image. Autopsy successfully extracted email messages, which can be examined for any potentially suspicious communications with external parties. Additionally, Autopsy retrieved details about web activity, including browsing history, downloads, and search data. Within the extracted sha256 directory, the contents of the image, including folders and files, are accessible for further inspection.

The initial step in the analysis involve focusing on the emails extracted by Autopsy. In this stage, the primary objective is to identify emails that could provide valuable insights into the incident, as well as to examine any messages sent to "unknown" recipients, those who are not affiliated with the company or its associated networks. The first email that draws attention is one received from Elliot Alderson on *October 24th at 10:19 UTC*, to which Sarah Williams, the user of the examined computer, agreeing to an invitation to discuss about some system changes.

Figure 6.2: First Suspicious Email



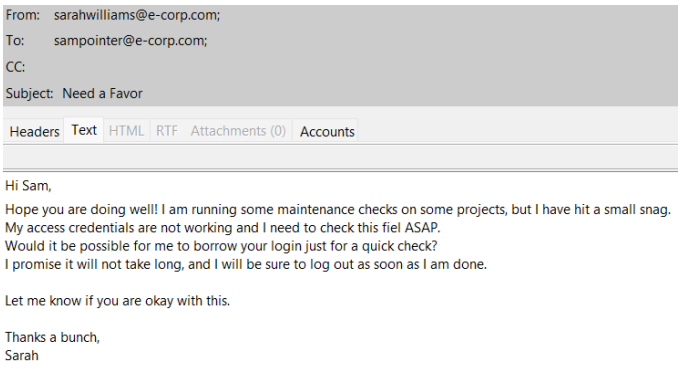
(a) Email Body



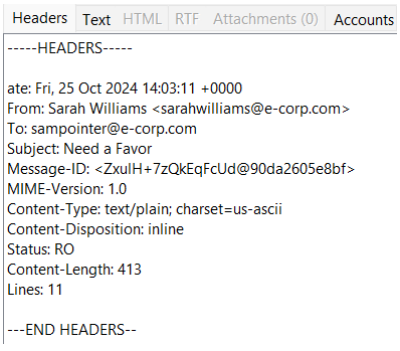
(b) Email Header

After identifying the initial suspicious email, the search continues for additional relevant communications. Among the emails sent and received by Sarah Williams, there is one sent to Sam Pointer on October *25th at 14:03 UTC*, asking if she can use his credentials due to issues with her account. Sam Pointer agrees but insists on providing the credentials in person to avoid sharing them over email. Upon discovering this email, it is then necessary to re-examine the database logs to determine whether this email exchange aligns with the use of Sam Pointer's credentials and if the dates correspond.

Figure 6.3: Sarah Williams asks to Sam Pointer his credentials



(a) Email Body

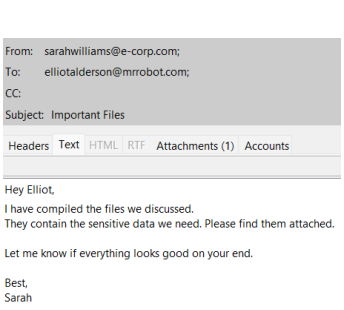


(b) Email Header

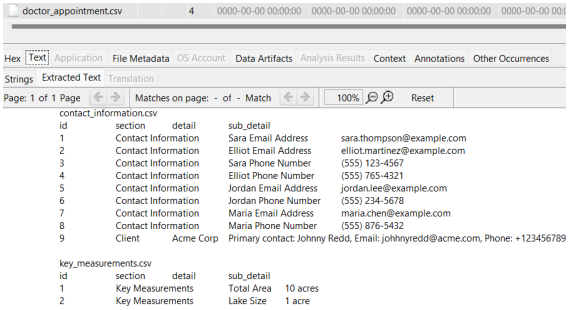
Given the date of Sarah's email (October 25th) and the timestamp of her initial interaction with the database server using Sam's credentials (October 26th), it can be deduced that Sarah utilized her colleague Sam's credentials to extract information from the database while attempting to avoid drawing attention to herself. Another interesting email is the one from Sarah Williams to Elliot with the subject line *Important Files* and an attachment. In this instance, Autopsy did not

detect the date of the email, so an effective way to estimate when it was sent is to examine any replies related to this message.

Figure 6.4: Sarah Williams sends a File to Elliot Alderson



(a) Email Body

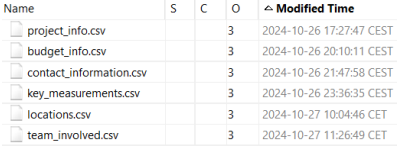


(b) doctor_appointment.csv

(c) Email Date

Based on Elliot's reply, it is possible to see that Sarah's email was sent on *October 27th*, the same day Elliot responded. In the top right corner of the Figure 6.4c, there is the date of the Elliot's response (*October 27th at 15:40 CET*), while in the body of the email there is the date of when the email was sent by Sarah (*October 27th at 12:13 UTC*). This timestamp follows the suspicious database accesses, suggesting that the content of the attachment deserves further analysis. Opening the attachment (Figure 6.4b), named *doctor_appointment.csv* likely to make it appear as a private document, reveals various pieces of data separated by file names, including information on company projects, associated costs, and team details. The next investigation step involves checking whether *doctor_appointment.csv* is still on Sarah's computer or not. However, a search of the folders extracted by Autopsy reveals only files containing data extracted from the database but not the emailed file, indicating that Sarah may have deleted the email attachment after sending it. The files were located in a folder named *personalStuff*, with the full path */home/sarahwilliams/personalStuff*. These files show modification dates between October 26th and October 27th, which align with the database access logs. It is important to note that the timestamps in the database logs are recorded in *UTC* (Coordinated Universal Time), while Autopsy reports times in both *CEST* (Central European Summer Time, UTC+2) and *CET* (Central European Time, UTC+1).

Figure 6.5: Leaked Files and Timestamps



Since the file sent via email is no longer on the computer, a possible next step is to examine the history of command-line operations. This command history is stored in the *.bash_history* file. To locate this file efficiently, the *Keyword Search* module in Autopsy can be used by entering the file name directly. Reviewing the command history reveals the creation of a folder named *doctorApp* inside the previously mentioned *personalStuff* folder. A few commands later, there is a sequence used to populate the *doctor_appointment.csv* file, followed by a command that deletes

the *doctorApp* folder. This sequence likely explains why the attached file is no longer present on the computer.

6.3.1 Result of the Investigation

In conclusion, this analysis suggests the involvement of Elliot Alderson as the primary attacker, Sarah Williams as his accomplice, and Sam Pointer as the unaware victim. The sequence of events begins with Elliot and Sarah coordinating over email, the first one sent by Elliot on October 24th inviting Sarah to meet and discuss certain system changes and how they should approach them (Figure 6.2). After several exchanges, on October 25th, Sarah contacts her colleague Sam Pointer (Figure 6.3), requesting access to the company database with his credentials under the pretense of encountering technical issues with her own. Trusting her, Sam agrees to provide his credentials in person rather than by email. Using Sam's credentials, Sarah accesses the database over the next two days, triggering suspicious activity alerts. The first access occurs on *October 26th at 15:27 UTC* (Listing 6.1), and the last on *October 27th at 10:26 UTC* (Listing 6.2). During these interactions, Sarah saves the extracted database information into several files (Figure 6.5) and subsequently merges their contents into a single file, *doctor_appointment.csv*, stored within a folder named doctorApp located at */home/sarahwilliams/personalStuff/doctorApp*. On *October 27th at 12:13 UTC*, Sarah emails this new file to Elliot (Figure 6.4). She then deletes the doctorApp folder, including its contents, in an apparent attempt to remove any remaining traces of her activity (ADD THE AUTO REF TO THE IMAGE THAT I NEED TO ADD FOR THE BASH HISTORY).

Chapter 7

Phishing Attack Simulation

7.1 Description of the Architecture

The architecture used to simulate a phishing attack consists of an external attacker, represented by Elliot Alderson, and a GoPhish server, which is used to design and launch the phishing campaign. In this simulated phishing attack, the attacker impersonates the company's IT team, sending an email that requests employees to enter their credentials to ensure database functionality during maintenance. The email directs recipients to a webpage hosted on the GoPhish server, where a form requests for usernames and passwords. Within the simulation, some employees submit their credentials, one employee opens the link without entering any information, and another employee ignores the email entirely. he attack did not end with the theft of credentials, but continued with Elliot using them to access the database and steal some data. To avoid suspicion, he alternated between the various credentials, distributing the database queries among multiple user accounts rather than relying on a single account for all database interactions. The company structure was developed according to the foundational elements outlined in the chapter on Docker setup (chapter 5), with additional containers introduced for the external attacker and the GoPhish server.
GoPhish is a powerful, open-source phishing framework designed to test an organization's exposure to phishing attacks. It facilitates the creation or import of highly realistic phishing templates, allowing the replication of authentic-looking emails and websites. GoPhish operates through a straightforward setup that provides a centralized interface for managing phishing campaigns. Its built-in editor enables administrators to design emails that closely resemble real-world phishing attempts, mimicking legitimate messages employees might receive from familiar services, colleagues, or clients. Once phishing emails are deployed, GoPhish monitors recipient interactions, tracking actions such as email opens, link clicks, and instances where users enter sensitive information on fake login pages. This tracking provides valuable insights into employee responses to phishing attempts, identifying vulnerabilities within the organization's security practices.

7.1.1 GoPhish Setup

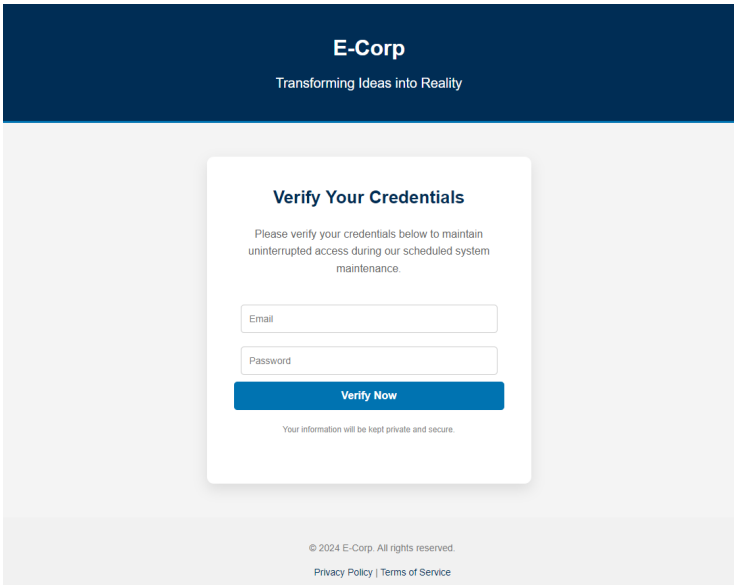
The initial configuration step involves setting up the **sending profile**, which includes the SMTP server details that GoPhish will use to send phishing emails. This requires entering information such as the server address, authentication credentials, and the email address to be displayed as the sender. Correctly configuring these settings is essential to ensure that the emails successfully reach the intended recipients. For email spoofing, the *SMTP From* parameter should specify an email address that differs from the sender's actual address.

Listing 7.1: Sending Profile Example

```
1 SMTP From: elliotalderson@mrrobot.com
2 SMTP From: itsupport@e-corp.com # For Email Spoofing
3 Host: email_server:587
4 Username: elliotalderson
5 Password: elliotalderson
```

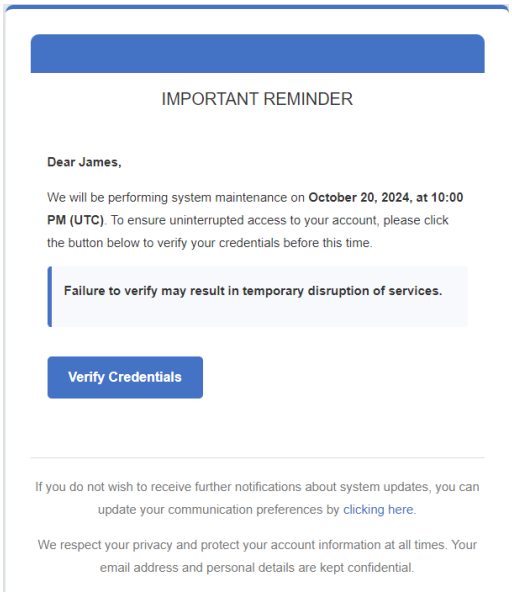
The **landing page** is the destination where targeted individuals are directed if they click on a link in the phishing email. This page can be a replicated version of a legitimate website's login page or another form designed to deceive users into providing sensitive information. GoPhish enables importing an HTML file or creating a page using its editor, allowing for tailored customization of the page's appearance and functionality. To capture data submitted on the page, selecting the checkbox at the bottom of the form creation section is necessary, with an additional checkbox available for capturing passwords. Additionally, users can be redirected to another website immediately after entering credentials by inserting the desired link into the *Redirect To* field.

Figure 7.1: Landing Page Example



The **email template** is crucial to the phishing campaign, as it is the crafted message that appears to come from a legitimate source. GoPhish includes an editor that allows users to design emails resembling real ones, such as a security alert from a well-known service or a routine internal request. To enhance credibility, the template can incorporate logos, specific layouts, and personalized fields, like the recipient's name, to make the message appear authentic. A common tactic in phishing emails is embedding a hyperlink that leads to a malicious website. GoPhish enables links to be inserted with anchor text, such as *Click here to update your account* or *Verify your identity*. To spoof the sender information, the *Envelope Sender* can be set to a name like *Elliot Alderson <elliotalderson@mrrobot.com>*, while modifying the visible content to a name like *IT TEAM <itsupport@e-corp.com>* enhances the email's legitimacy and believability.

Figure 7.2: Email Template Example



Listing 7.2: Email Template Example

```
1 Envelope Sender: Elliot Alderson <elliotalderson@mrrobot.com>
2 Envelope Sender: IT TEAM <itsupport@e-corp.com> # Spoof
3 Subject: Scheduled System Maintenance -Verify Your Credentials
```

Once the email and landing page are configured, the next step involves defining the target audience, **Users & Groups**, by creating a recipient group with names and email addresses. This can be done manually or by importing a larger list via a CSV file. Careful selection of the target audience ensures that the campaign reaches the intended recipients, providing accurate data on how various groups respond to the phishing attempt.

Table 7.1: Users & Groups Example

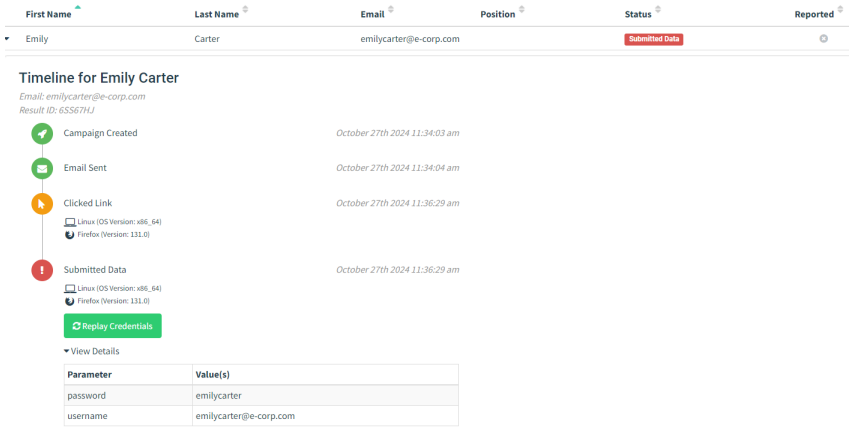
First Name	**Last Name**	**Email**
Sarah	Williams	sarahwilliams@e-corp.com
James	Foster	jamesfoster@e-corp.com
Emily	Carter	emilycarter@e-corp.com
John	Doe	johndoe@e-corp.com
Sam	Pointer	sampointer@e-corp.com

With all components prepared, the **campaign** can be launched by assigning it a name, selecting the designated email templates and landing pages, and setting the SMTP profile. The emails can be scheduled for immediate release or set to send at a specific time. Once active, GoPhish begins distributing the phishing emails to recipients and tracking their interactions in real time.

7.1.2 GoPhish Campaign

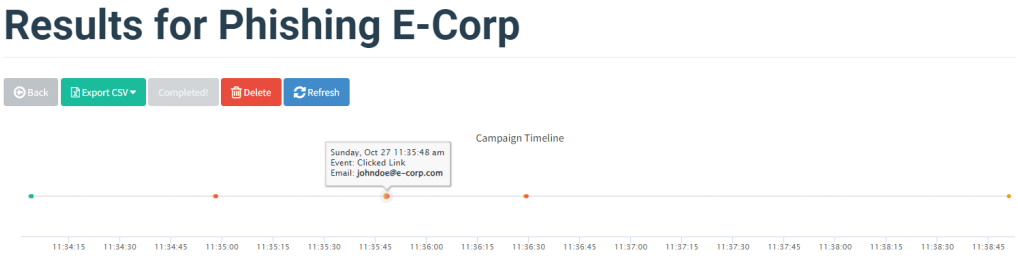
Figure 7.3: GoPhish Campaign Overview

(a) Clicked Link



(b) Data Submission

(c) Campaign Dashboard



(d) Campaign Timeline

7.2 How to Detect a Phishing Attack

Detecting a phishing attack can be achieved in several ways that can also help reduce the consequences of it. **User Behavior Analytics** (UBA) is an approach used to detect phishing attacks. These attacks often aim to gain unauthorized access to an organization's network, typically through stolen login credentials or malware. Behavioral analysis focuses on identifying unusual or suspicious user actions that might indicate a phishing related breach. For example, an unusual request to extract large volumes of data from the company's database could signal a data breach using compromised credentials. **URL filtering** is a critical method in detecting phishing attacks, as phishing emails frequently include links to fake websites designed to steal login credentials or deploy malware. URL filtering examines the links within an email to identify malicious or suspicious URLs. This includes checking for known harmful domains, look-alike URLs, or links whose structure suggests potential threats. There are also **email scanning** solutions, which analyze the contents and attachments of emails for signs of phishing. These tools use various techniques, including *sandboxing*, where attachments are opened in a secure environment to detect any harmful behavior, such as malware execution. In addition, **artificial intelligence** (AI) and **machine learning** (ML) technologies are increasingly being utilized to detect phishing attempts by analyzing email content and the websites linked within them. These AI models are trained to recognize typical signs of phishing, such as spelling errors, urgent language intended to pressure the recipient, and suspicious URL structures. By assessing these factors, AI can determine the likelihood of an email being a phishing attempt, enabling it to block potentially harmful emails before they reach the recipient's inbox [45].
To protect a company from phishing attacks, **email security tools** are essential. These tools, including spam filters, anti-virus software, and multi-factor authentication (MFA), work together to block phishing attempts before they reach employees' inboxes. However, email security tools alone may not provide complete protection. Employees must also be educated on how to recognize phishing attempts and properly handle suspicious emails or requests for sensitive information. **User awareness** training is a crucial element in any anti-phishing strategy. Employees need to be trained to recognize phishing attempts, which may involve recognizing suspicious emails or requests for sensitive information. Regular training sessions ensure employees remain informed about the latest phishing tactics and are prepared to respond appropriately. In addition, the training should emphasize best practices in password management and safe online browsing to reduce the likelihood of falling victim to phishing schemes. Educating employees helps organizations mitigate the risk of phishing and protects sensitive data from unauthorized access. It is also important to implement **multi-factor authentication** (MFA), because it adds an additional layer of security to the login process, making it harder for attackers to gain unauthorized access, even if they have compromised login credentials. By requiring users to provide multiple forms of identification, such as a password and a code sent to their phone, MFA significantly enhances security [46].

7.3 How to Investigate a Phishing Attack

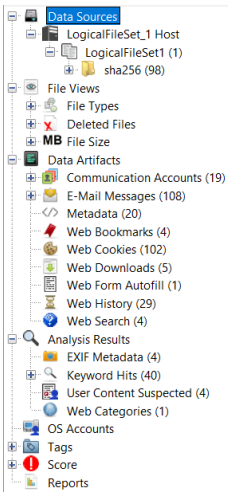
When a report of a potential phishing attack is received, possibly flagged by an email analysis tool, a database activity monitoring tool, or a well trained employee who identified and reported a suspicious email, the next step is to determine which users received the malicious email and how they reacted, whether they followed the email's instructions or recognized it as fraudulent.
The initial step in the investigation involves analyzing the email server image to locate the suspicious email and identify its recipients. In this case, the email server logs (Listing 7.3) show that the email was received by employees on *October 29 at 09:07 UTC*. Among the details that suggest the suspicious nature of the email, there are the username used for authentication (*elliotalderson*) and the IP address (*203.0.113.3*), external and unknown to the company, from which the sender connected and authenticated. The example logs provided (Listing 7.3) show only the first two employees that received the email.

Listing 7.3: Email Server Logs

```
1  Oct 29 09:07:40 e-corp postfix/submission/smtpd[4792]: 7428BC7C3D7:
       client=unknown[203.0.113.3], sasl_method=PLAIN, sasl_username=elliotalderson
2  Oct 29 09:07:40 e-corp postfix/cleanup[4797]: 7428BC7C3D7:
       message-id=<17301928604734950421.1.27929103682845359 47@f04f9ce70673>
3  Oct 29 09:07:40 e-corp postfix/qmgr[4457]: 7428BC7C3D7: from=<itsupport@e-corp.com>,
       size=4577, nrcpt=1 (queue active)
4  Oct 29 09:07:40 e-corp postfix/local[4798]: 7428BC7C3D7: to=<sarahwilliams@e-corp.com>,
       relay=local, delay=0.02, delays=0/0/0/0.01, dsn=2.0.0, status=sent (delivered to mailbox)
5  Oct 29 09:07:40 e-corp postfix/qmgr[4457]: 7428BC7C3D7: removed
6
7  Oct 29 09:07:40 e-corp postfix/submission/smtpd[4792]: 6F680C7C3D7:
       client=unknown[203.0.113.3], sasl_method=PLAIN, sasl_username=elliotalderson
8  Oct 29 09:07:40 e-corp postfix/cleanup[4797]: 6F680C7C3D7:
       message-id=<17301928604531635631.1.8376357689856336571@f04f9ce70673>
9  Oct 29 09:07:40 e-corp postfix/qmgr[4457]: 6F680C7C3D7: from=<itsupport@e-corp.com>,
       size=4566, nrcpt=1 (queue active)
10 Oct 29 09:07:40 e-corp postfix/local[4799]: 6F680C7C3D7: to=<sampointer@e-corp.com>,
       relay=local, delay=0.02, delays=0.01/0/0/0.01, dsn=2.0.0, status=sent (delivered to mailbox)
11 Oct 29 09:07:40 e-corp postfix/qmgr[4457]: 6F680C7C3D7: removed
12 ...
```

After identifying potential victims, the investigation then shifts to analyzing the devices of users who received the email to determine their actions. The investigation on the victim computer is performed with Autopsy. After the image is loaded into Autopsy, the *Embedded File Extraction* module is run, and after its successful execution, additional modules can be applied. In this case, the *Recent Activity* and *Email Parser* modules are particularly useful because they extract information like sent and received emails, and web history.

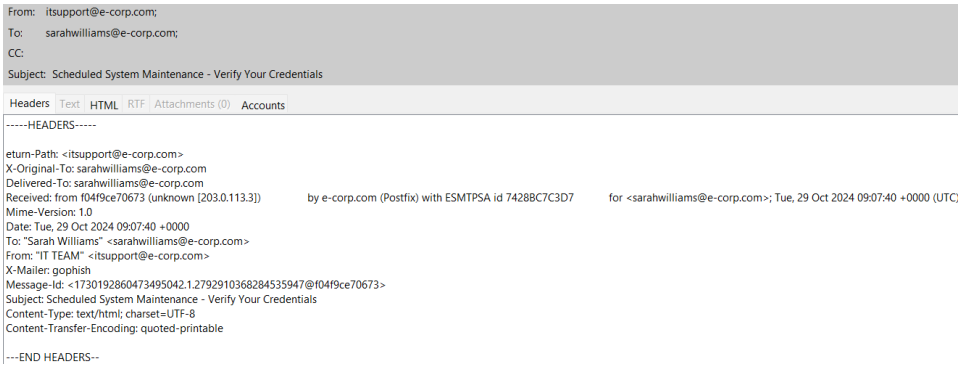
Figure 7.4: Result Run Ingest Modules in Autopsy



Among emails received on that date, one stands out with the sender *"IT TEAM" <itsupport@e-corp.com>* and a subject, "*Scheduled System Maintenance - Verify Your Credentials*", intended to draw attention and appear more official. The email can be identified by looking either at the date and time of receipt or at the *Message-Id*, which appears both in the email server logs and in the header of the email stored in the victim computer. The attacker's objective is to create an email that appears authentic and professional to deceive the employees. In this simulation, the final version of the email is displayed in Figure 7.2.

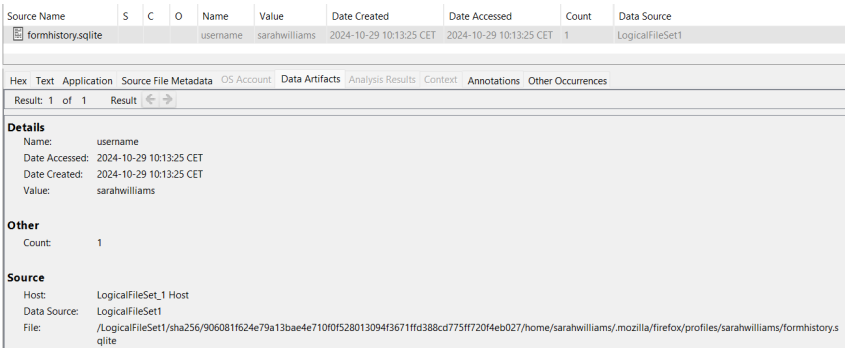
After identifying the email corresponding to the suspicious entry in the logs, the next step is to investigate the actions taken by the user on the computer in question. This involves determining whether the user ignored the email or opened it and followed its instructions. One effective approach is to analyze the user's internet activity, as phishing emails of this type, as seen in this case, often contain links, such as those asking users to verify their credentials.

Figure 7.5: Phishing Email Header



In Autopsy, the left-side panel (shown in the Figure 7.4) displays the results of modules that have extracted different information from the computer, including browser activities. By selecting *Web History*, it shows a list of sites the user has visited. Although no suspicious sites appear in the browsing history, the results panel also includes a *Web Form Autofill* section, which shows that credentials were entered on a site on *October 29 at 10:13 CET* (UTC+1).

Figure 7.6: Web Form Autofill



However, the site associated with that timestamp does not appear in the browsing history, so a manual review of the *places.sqlite* file, where Firefox stores internet browsing data, is necessary. Within *places.sqlite*, the *moz_origins* table (Figure 7.7a) contains an entry with ID 25, *http://* as a prefix and an IP address (*203.0.113.3*) as a host, which matches the IP from which the suspicious email originated. Additionally, the *moz_places* table (Figure 7.7b) indicates that the user visited a website containing this IP in the URL, making it suspicious. In this case, Autopsy records this timestamp in milliseconds, which, once converted, confirms the exact date and time of the visit: October 29th at 09:13 UTC. In this case, the date and time of the visit to the suspicious website and the timestamp from the *Web Form Autofill* section align, allowing the conclusion that the user, *sarahwilliams*, was deceived by the attacker, leading to the compromise of her credentials. This analysis should be repeated for all users who were identified as having received the suspicious email in order to gain a comprehensive understanding of the impact of the attack.
Once all affected users have been examined, the next step is to review the database logs to determine whether the compromised credentials have been used by an external party. A detailed examination of the logs (example in Listing 7.4), with particular focus on the users who entered their credentials in the fake login screen, reveals several database operations conducted from an IP address (*203.0.113.2*) that is unusual for the affected users. From the logs it is possible to notice that this IP address appears only after the phishing attack and that it makes several interactions with the database using the stolen credentials to extract data relating to internal projects of the company.

Figure 7.7: Tables in places.sqlite

Table moz_origins 25 entries Page 1 of 1 Export to CSV

id	prefix	host	frecency	recalc_fr...	alt_frece...	recalc_alt...
21	https://	workspace.google.com	136	0		1
22	https://	www.ukbumpkeys.com	100	0		1
23	https://	www.imperva.com	0	0		1
24	https://	www.majowiecki.com	0	0		1
25	http://	203.0.113.3	100	0		1
3	https://	killedbygoogle.com	100	0		1
4	https://	www.google.com	800	0		1
5	https://	www.accuweather.com	100	0		1

(a) Unique origins (moz_origins)

Table moz_places 31 entries Page 1 of 1 Export to CSV

id	url	title	rev_host	visit_count	last_visit_date
27	https://workspace.google.com/in..	Shareable Online Calendar and .	moc.elgoog....	4	1730556466742610
28	https://www.ukbumpkeys.com/c...	Shop Lock Pick Sets - #1 UK lo...	moc.syekpm...	1	1730292320784115
29	https://www.imperva.com/resour...	More-Lessons-Learned-from-A...	moc.avrepmi...	0	1730177512586000
3	https://www.mozilla.org/contribu..		gro.allizom....	0	
30	https://www.majowiecki.com/pdf..	MJW-structures-portfolio.pdf	moc.ikceiwoj...	0	1730171512567000
31	http://203.0.113.3/?rid=T2FM6e9	Verify Your Credentials	3.311.0.302.	1	1730193205687857
4	https://www.mozilla.org/about/		gro.allizom....	0	
5	https://killedbygoogle.com/	Google Graveyard - Killed by G..	moc.elgoogy..	1	1730284666209739

(b) Visited URLs (moz_places)

Listing 7.4: Log Example Phishing Attack

```
1 2024-10-30 13:35:26.532 UTC [750]: user=[unknown], db=[unknown], client=203.0.113.2 LOG:
      connection received: host=203.0.113.2 port=59898
2 2024-10-30 13:35:26.545 UTC [750]: user=sarahwilliams, db=evil_corp, client=203.0.113.2 LOG:
      connection authenticated: identity="sarahwilliams" method=md5
      (/etc/postgresql/16/main/pg_hba.conf:25)
3 2024-10-30 13:35:26.545 UTC [750]: user=sarahwilliams, db=evil_corp, client=203.0.113.2 LOG:
      connection authorized: user=sarahwilliams database=evil_corp application_name=psql SSL
      enabled (protocol=TLSv1.3, cipher=TLS_AES_256_GCM_SHA384, bits=256)
4 2024-10-30 13:35:26.547 UTC [750]: user=sarahwilliams, db=evil_corp, client=203.0.113.2 LOG:
      statement: SELECT * FROM contact_information;
5 2024-10-30 13:35:26.549 UTC [750]: user=sarahwilliams, db=evil_corp, client=203.0.113.2 LOG:
      duration: 0.452 ms
6 2024-10-30 13:35:26.549 UTC [750]: user=sarahwilliams, db=evil_corp, client=203.0.113.2 LOG:
      disconnection: session time: 0:00:00.047 user=sarahwilliams database=evil_corp
      host=203.0.113.2 port=59898
```

7.3.1 Result of the Investigation

In conclusion, the investigation conducted in response to the phishing attack has clarified both the affected users and the sequence of events. An external attacker launched a phishing campaign using the tool GoPhish (as indicated in the email header in the X-Mailer field, shown in Figure 7.5), posing as the IT team. The attacker warned employees of scheduled system maintenance, requesting that they insert their credentials to maintain access to database-related services (email example in Figure 7.2). This message directed users to a fake login page designed to capture any entered credentials (fake login form in Figure 7.1).
The investigation revealed that *sarahwilliams* entered her credentials into this fake login form, as Autopsy's Recent Activity module detected the username entry (Figure 7.6), and the timestamps match the site visit recorded (Figure 7.7b). Additionally, since the attack, the database logs have shown access attempts under Sarah Williams's account (and other employees' accounts) from an external IP address, *203.0.113.2* (Listing 7.4). The analysis process applied to Sarah Williams's computer was also conducted for all users who received the phishing email, providing a comprehensive overview of the attack's impact. The campaign's results from the attacker's perspective are illustrated in Figure 7.3.

Chapter 8

Ransomware Attack Simulation

8.1 Description of the Architecture

Starting from the basic configuration discussed in chapter 5, an external attacker, identified as Elliot Alderson, is introduced. This attacker uses a spoofed email address to impersonate a member of the IT team, and sends an urgent email instructing employees to install a critical update. The attached zip file in this message is, in fact, a ransomware. Once installed, this malicious software specifically targets designated folders on the victim's computer, encrypting their contents to prevent access. The behavior of the ransomware has been replicated using Python scripts. The zip file sent via email contains an installer script, an update folder with the malicious code, a readme with instructions for installing the update, and a requirements.txt file listing necessary packages. The *installer.py* script first identifies the current execution path, installs the required packages from the requirements.txt file, and then moves the malicious code to a specific directory (*/usr/local/bin/update/*). The final action of the installer script is to execute the malicious *update.py* script. The *update.py* script serves as the core of the ransomware. It systematically targets the home directories of each user on the computer by retrieving a list of active user accounts and attacking all folders within paths structured as */home/username/*. Additionally, the directory containing the *update.py* script itself (*/usr/local/bin/update/*) is also targeted, ensuring that the ransomware code is encrypted alongside other data, in order to complicate any forensic investigation. After encrypting these folders, the script generates a ransom note demanding payment to regain access to the compromised files.

8.2 How to Detect a Ransomware Attack

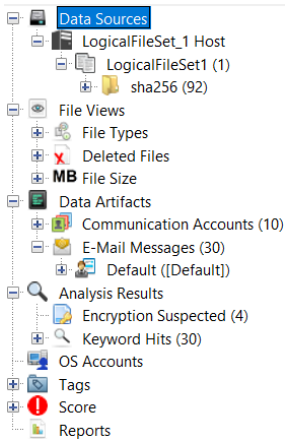
Ransomware attacks can be detected through various approaches, each with unique strengths and limitations. One common technique, known as **signature-based detection**, identifies ransomware by comparing file hashes to a database of known ransomware signatures, allowing for rapid file analysis. However, this approach has limitations, as it is ineffective against newly developed ransomware variants; attackers can easily alter malware files to create new hashes, which evade detection. Minor file modifications, such as the addition of a single byte, can generate a unique hash that bypasses this type of detection. Another approach to ransomware detection involves **analyzing network traffic**, focusing on data exchanged between endpoints. This method examines time stamps, data volumes, and other metrics to identify unusual patterns that may suggest a ransomware attack. When suspicious activity is detected, the system can initiate a lockdown to prevent further damage. A significant advantage of network traffic analysis is its independence from ransomware signatures, allowing it to detect unknown ransomware. However, this approach may generate false positives, mistakenly blocking legitimate files. **Monitoring file storage behaviors** offers another layer of ransomware detection. Ransomware typically encrypts or locks files before issuing a decryption ransom demand, so tracking unexpected shifts in file storage locations or detecting sudden spikes in file encryption activity can signal an ongoing

attack. This method does not rely on known malware patterns, which reduces the likelihood of false positives [47]. The use of **honeypot folders** presents another low-cost, proactive strategy in ransomware detection. Honeypots are decoy folders, strategically named and hidden to attract ransomware. These folders are designed to contain files that would appear interesting to ransomware but would be ignored by legitimate users. Although there is no guarantee that ransomware will attack these folders first or respect their hidden status, this method offers an inexpensive and straightforward way to detect ransomware behavior, though it may become less effective as ransomware creators become more sophisticated [48]. Ransomware may also target specific files whose encryption disrupts the functionality of certain programs. In the simulation conducted in this thesis, for example, the ransomware encrypts the muttrc file, which is essential for configuring access parameters for the email client mutt. Once this configuration file is encrypted, the system experiences access issues with mutt, demonstrating how ransomware can create significant operational disruptions by targeting critical configuration files.

8.3 How to Investigate a Ransomware Attack

When a ransomware attack is detected, whether through a user reporting issues with their computer, the discovery of a ransom note, or a tool identifying anomalous file encryption activities, it becomes essential to isolate the affected systems immediately. This isolation allows for a thorough analysis to determine the nature of the attack, the methods employed, and the resulting impact. Understanding the sequence of events and the extent of the damage is critical for both mitigating the immediate threat and preventing future incidents.
To analyze the devices reported as infected by a ransomware, the first step involves using Autopsy to examine the data. After importing the image into Autopsy and running the necessary modules, it becomes evident that the information extracted from this case is less comprehensive compared to previously analyzed cases (see Figure 6.1 and Figure 7.4).

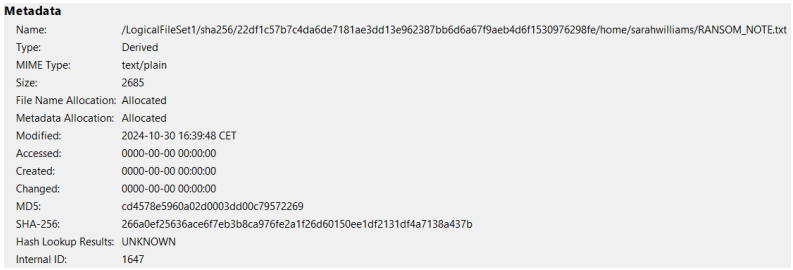
Figure 8.1: Result Run Ingest Modules in Autopsy



Looking the the results of the ingest modules, there is an absence of internet browsing history and a lower number of extracted emails. The inability of Autopsy to extract the emails received by the user is due to the encryption of the inbox file, as illustrated in the Figure 8.3. From this image, it is possible to see the date and time of the last modification to the inbox file, recorded as *October 30th at 16:39 CET* (UTC+1). This timestamp provides valuable information for constructing a timeline of the attack's progression. A similar issue occurred with files containing information related to internet browsing, which were also encrypted. The lack of web browsing data complicates the reconstruction of events, as it hinders the ability to determine the online activity of the user and whether any suspicious downloads occurred. Regarding the emails, the extracted content consists exclusively of sent emails, with no received emails available for review. While navigating through the analyzed device, a file named *RANSOM_NOTE.TXT* was discovered in the directory */home/sarahwilliams/* (Figure 8.2). This file, created on *October 30th at 16:39 CET* (UTC+1), indicates that the contents of the computer have been encrypted and demands

payment for recovery. This discovery provides a potential timeline for the ransomware attack, pinpointing the event to *October 30th at 15:39 UTC*.

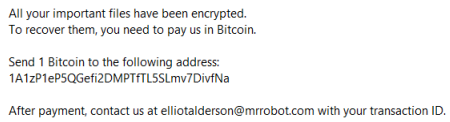
Figure 8.2: Ransom note



Metadata

Name:	/LogicalFileSet1/sha256/22df1c57b7c4da6de7181ae3dd13e962387bb6d6a67f9aeb4d6f1530976298fe/home/sarahwilliams/RANSOM_NOTE.txt
Type:	Derived
MIME Type:	text/plain
Size:	2685
File Name Allocation:	Allocated
Metadata Allocation:	Allocated
Modified:	2024-10-30 16:39:48 CET
Accessed:	0000-00-00 00:00:00
Created:	0000-00-00 00:00:00
Changed:	0000-00-00 00:00:00
MD5:	cd4578e5960a02d0003dd00c79572269
SHA-256:	266a0ef25636ace6f7eb3b8ca976fe2a1f26d60150ee1df2131df4a7138a437b
Hash Lookup Results:	UNKNOWN
Internal ID:	1647

(a) Ransom Note Metadata



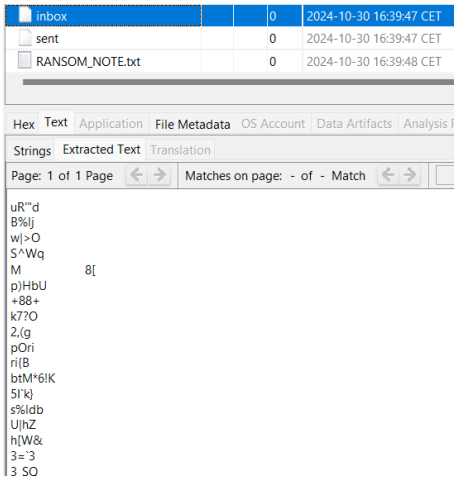
All your important files have been encrypted.
To recover them, you need to pay us in Bitcoin.

Send 1 Bitcoin to the following address:
1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa

After payment, contact us at elliotalderson@mrrobot.com with your transaction ID.

(b) Content Ransom Note

Figure 8.3: Inbox File Encrypted



inbox		0	2024-10-30 16:39:47 CET
sent		0	2024-10-30 16:39:47 CET
RANSOM_NOTE.txt		0	2024-10-30 16:39:48 CET

Hex | Text | Application | File Metadata | OS Account | Data Artifacts | Analysis

Strings | Extracted Text | Translation

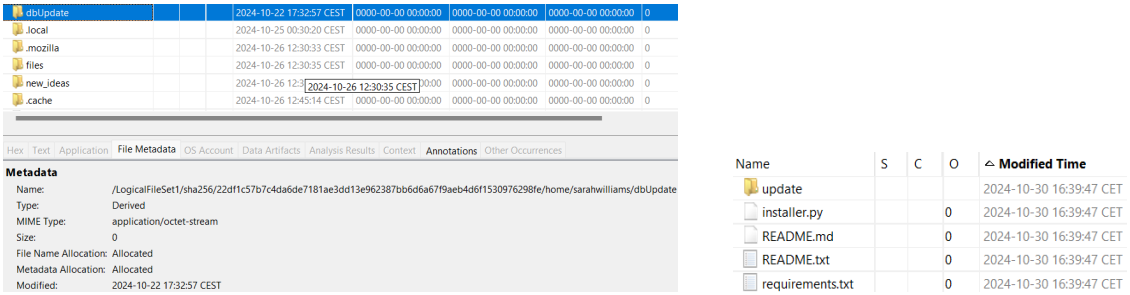
Page: 1 of 1 Page | Matches on page: - of - Match

```
uR'"d
B%lj
w|>O
S^Wq
M                8[
p)HbU
+88+
k7?O
2,(g
pOri
ri{B
btM*6!K
5I`k}
s%ldb
U|hZ
h[W&
3=`3
3 SO
```

Within the same directory, a folder named *dbUpdate* was identified, while a file named *dbUpdate.zip* was located in */home/sarahwilliams/download/*. The timestamp of the folder, *October 30th at 17:32 CEST* (UTC+2), suggests it was extracted approximately seven minutes before the encryption of all files in */home/sarahwilliams/* (Figure 8.4a). However, the contents of the dbUpdate folder were also encrypted, so it is hard to understand what it contains (Figure 8.4b). Determining the origin of this folder is essential, as it may represent the source of the attack. Tracing its origins provides critical insights into the events leading up to the incident. The zip file could have been introduced to the victim's computer through various means, such as an email attachment or an internet download. Identifying the method of delivery is key to reconstructing the sequence of events and understanding the progression of the attack. To continue the investigation, the logs of the email server were analyzed to identify any emails containing suspicious attachments or links sent during the relevant time frame. The logs revealed that on *October 30th at 11:50 UTC*, an email was sent from *IT Team <itsupport@e-corp.com>*, with the IP address *203.0.113.2*, to four different employees. This email was addressed to multiple recipients, including Sarah Williams, the owner of the analyzed device (Listing 8.1). After identifying a series of emails in the logs that may have been used to deliver the ransomware to various employees, it becomes necessary to analyze the image of the email server. Using Autopsy, specifically the *Email Parser* module, all emails exchanged between the employees' accounts are extracted. This process

enables the identification of the suspicious email, allowing for further investigation to determine if it is connected to the ransomware or if additional avenues need to be explored to locate the source of the attack. The email invited the recipients to install an update as soon as possible, and the attachment, *dbUpdate.zip*, had the same name as the folder found, and mentioned earlier, on the victim's device (Figure 8.5). This compressed folder was extracted, as shown by the timestamps, shortly before some of the files on the computer were encrypted by the ransomware.

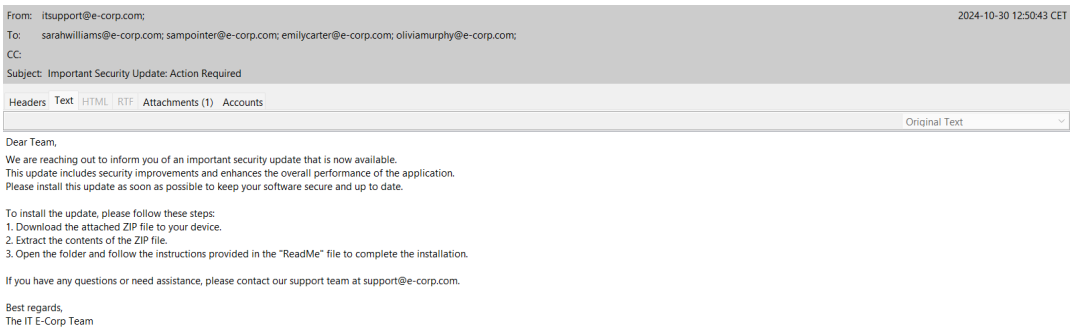
Figure 8.4: Email Attachment



(a) Metadata dbUpdate Folder

(b) Content dbUpdate Folder

Figure 8.5: Suspicious Email

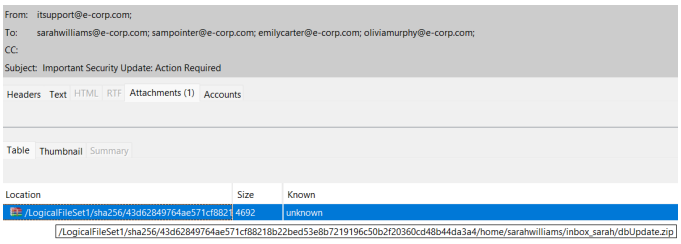


Listing 8.1: Email Server Logs

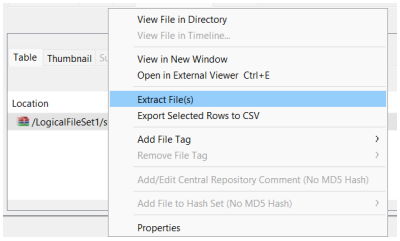
```
1  Oct 30 11:50:43 e-corp postfix/submission/smtpd[5471]: connect from unknown[203.0.113.2]
2  Oct 30 11:50:43 e-corp postfix/submission/smtpd[5471]: 91DA0BD0AD2:
       client=unknown[203.0.113.2], sasl_method=PLAIN, sasl_username=elliotalderson
3  Oct 30 11:50:43 e-corp postfix/cleanup[5475]: 91DA0BD0AD2:
       message-id=<ZyIdk6Udp7gvNqJd@4da43bbde52e>
4  Oct 30 11:50:43 e-corp postfix/qmgr[5424]: 91DA0BD0AD2: from=<itsupport@e-corp.com>,
       size=7943, nrcpt=4 (queue active)
5  Oct 30 11:50:43 e-corp postfix/submission/smtpd[5471]: disconnect from unknown[203.0.113.2]
       ehlo=2 starttls=1 auth=1 mail=1 rcpt=4 data=1 quit=1 commands=11
6  Oct 30 11:50:43 e-corp postfix/local[5476]: 91DA0BD0AD2: to=<emilycarter@e-corp.com>,
       relay=local, delay=0.15, delays=0.09/0.02/0/0.04, dsn=2.0.0, status=sent (delivered to
       mailbox)
7  Oct 30 11:50:43 e-corp postfix/local[5477]: 91DA0BD0AD2: to=<oliviamurphy@e-corp.com>,
       relay=local, delay=0.18, delays=0.09/0.04/0/0.05, dsn=2.0.0, status=sent (delivered to
       mailbox)
8  Oct 30 11:50:43 e-corp postfix/local[5476]: 91DA0BD0AD2: to=<sampointer@e-corp.com>,
       relay=local, delay=0.19, delays=0.09/0.06/0/0.04, dsn=2.0.0, status=sent (delivered to
       mailbox)
9  Oct 30 11:50:43 e-corp postfix/local[5478]: 91DA0BD0AD2: to=<sarahwilliams@e-corp.com>,
       relay=local, delay=0.2, delays=0.09/0.07/0/0.04, dsn=2.0.0, status=sent (delivered to mailbox)
10 Oct 30 11:50:43 e-corp postfix/qmgr[5424]: 91DA0BD0AD2: removed
```

Using Autopsy, the *dbUpdate.zip* file was located and extracted for further analysis by right-clicking and selecting the *Extract File(s)* option (Figure 8.6). The subsequent step involves examining the contents of this attachment to determine if it served as the vector for the ransomware attack.
After extracting the potential ransomware, its contents can be subjected to analysis. The folder, as shown in the Figure 8.4b, includes several components: an installer.py script, a requirements.txt file, a readme file, and an update folder. The *requirements.txt* file specifies two packages, paramiko and cryptography, while the *readme* file provides instructions, directing the user to execute the installer.py script using the *python3 installer.py* command. The instructions describe the process as a critical security upgrade that introduces new features. In the end, inside the update folder there is a script called update.py. Initial analysis of the *installer.py* script reveals already suspicious behavior. The script installs the packages listed in requirements.txt, then relocates the contents of the update folder to a more hidden directory on the victim's computer. It subsequently executes the relocated script. Further examination of the *update.py* script demonstrates that its true function diverges significantly from the claims in the readme. Instead of enhancing security or functionality, its primary objective is to encrypt the files on the device where it is executed.

Figure 8.6: Email Attachment Extraction



(a) Suspicious Email Attachment



(b) Extract File From Autopsy

8.3.1 Result of the Investigation

At the conclusion of the investigation, it becomes possible to reconstruct the sequence of events and establish a timeline of the attack. The incident begins with an external attacker (elliotalderson), impersonating a member of the IT Team, who sends an email on *October 30th at 11:50 UTC* to several employees (Listing 8.1). The email (Figure 8.5) invites recipients to install an urgent update (Figure 8.6a), which is attached and contains a malicious Python script designed to encrypt the contents of specific folders on the victims' devices (Figure 8.4b). In the case of Sarah Williams' computer, analyzed in this section of the thesis using Autopsy, the zipped folder containing the fraudulent update was extracted on *October 30th at 15:32 UTC* (Figure 8.4a). The installation of the fake update appears to have occurred approximately seven minutes later, at *15:39 UTC*, based on the timestamp of the last modification to encrypted files (Figure 8.4b), such as the inbox file (Figure 8.3). This is also the time when the ransom note first appeared (Figure 8.2, marking the complete execution of the ransomware attack. In this kind of situations, It is crucial to emphasize the importance of avoiding compliance with the instructions provided in the ransom note. Instead, the appropriate course of action involves promptly contacting the relevant authorities and relying on backups, only if they have been properly maintained and are available.

Chapter 9

Malware Attack Simulation

9.1 Description of the Architecture

Starting from the basic configuration discussed in chapter 5, an external attacker, Elliot Alderson, is introduced to simulate a targeted social engineering attack. This attacker uses a spoofed email address to impersonate an IT team member, sending employees an urgent message that inviting them to install a critical update. Attached to the email there is a zip file containing malware disguised as the update. Once installed, this malicious code begins periodically transmitting the contents of specified folders to the attacker. The behavior of this malware is simulated through Python scripts that recreate the attack process. The fake update folder is designed to look legitimate, containing a *README* file with instructions on how to install the update, a *requirements.txt* file listing the packages needed to execute the script, an installer script, and an update folder that contains the fake update. The *installer.py* script initiates the attack by locating the directory where it resides, then using this path to install the required packages specified in *requirements.txt*. It checks whether *crontab*, a scheduling tool, is installed; if not, it installs it. To hide its operation, the script then moves the *update.py* file to */usr/local/bin/update/*. After identifying all user accounts on the target computer, the installer script adds *update.py* to crontab, specifying intervals for execution, and then,in the end, it starts crontab service. The *update.py* performs data exfiltration by sending the contents of designated target folders to a remote directory via SSH. For each user on the computer, the script creates a directory with the name of the user, and within each of these directories, it generates timestamped folders to store transferred files in an organized structure. Each time the script executes, a new timestamped folder is created, resulting in multiple folders over time that represent individual users. Within these user-specific directories, each timestamped folder is labeled with the corresponding date and time, effectively organizing the transferred files chronologically. This simulated attack scenario illustrates a sophisticated combination of social engineering and automated data theft, demonstrating how an attacker could exploit system vulnerabilities and user trust to gain unauthorized access to sensitive data.

9.2 How to Detect a Malware Attack

Various methods can be employed to detect a malware attack, each one focusing on identifying distinctive behaviors or anomalies in system performance and access patterns. **Endpoint behavior monitoring** involves observing processes running on endpoints for unusual activities, such as hidden or background services accessing sensitive folders. Signs of persistent malware may include changes to startup items, the addition of scheduled tasks, or registry modifications, which are tactics commonly used by malware to maintain access even after a reboot. In the case of this simulation, a new scheduled task has been added. Tracking **data access patterns** can also help identify potential attacks. Monitoring access to high sensitivity folders or files, particularly by unknown processes or users outside regular hours, can reveal unusual or unauthorized activity.

An increase in the frequency or timing of access to specific files may serve as an indicator of malicious behavior. **Network traffic analysis** is another effective approach, as many types of exfiltration malware communicate with external servers to transfer stolen data. Monitoring outbound traffic for unusual patterns, such as data being sent over non-standard ports or to unknown IP addresses, can highlight potential threats. Alerts for unexpected data uploads, along with the use of *Data Loss Prevention* (DLP) tools or *Intrusion Detection/Prevention Systems* (IDS/IPS), can assist in identifying and responding to suspicious data movement. **Malware honeypots** or **honeypot file**s are also valuable tools for detection. A *malware honeypot* emulates a software application or API to attract malware, allowing security teams to observe the behavior of the malware in a controlled environment. Similarly, *honeypot files* act as decoy files to attract and identify attackers. This approach allows security teams to analyze the specific techniques used in the attack, providing insights to improve anti malware defenses and address identified vulnerabilities [49]. Finally, a significant **decline in system performance** may indicate malware activity. Symptoms such as sudden disk space reduction, slow device response times, frequent system crashes or freezes, unexpected changes to browser settings, and redirected URLs can all point to an underlying malware presence.

9.3 How to Investigate a Malware Attack

Within a company, various methods can be employed to detect the potential presence of malware on devices. For example, in this simulation, malware detection could originate from a user reporting performance issues on their computer or from analyzing network traffic. By examining network traffic captured by the firewall using Wireshark, it is possible to see encrypted packet exchanges between the firewall and an external IP address. In this case, the IP address of the firewall appears due to NAT (Network Address Translation), revealing the IP address of the firewall interface instead of the one of the device in the inside network. There are multiple instances of these communications, such as the one observed on *November 17th at 14:02 UTC*, as shown in the Figure 9.1. The challenge then becomes determining the source and cause of these suspicious communications and which devices are involved.

Figure 9.1: Wireshark Capture on the Firewall

No.	Time	Source	Destination	Protocol	Length Info
836	120.624240076	198.51.100.1	203.0.113.2	TCP	66 58946 → 22 [ACK] Seq=1745 Ack=2795 Win=64128 Len=0 TSval=366100167 TSecr=428358385
837	120.624536994	198.51.100.1	203.0.113.2	SSHv2	130 Client: Encrypted packet (len=64)
838	120.624588998	198.51.100.1	203.0.113.2	SSHv2	114 Client: Encrypted packet (len=48)
839	120.632538585	203.0.113.2	198.51.100.1	SSHv2	450 Server: Encrypted packet (len=384)
840	120.633785162	198.51.100.1	203.0.113.2	SSHv2	194 Client: Encrypted packet (len=128)
841	120.634911131	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
842	120.635849088	198.51.100.1	203.0.113.2	SSHv2	194 Client: Encrypted packet (len=128)
843	120.636927454	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
844	120.637582795	198.51.100.1	203.0.113.2	SSHv2	210 Client: Encrypted packet (len=144)
845	120.639688824	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
846	120.640530575	198.51.100.1	203.0.113.2	SSHv2	210 Client: Encrypted packet (len=144)
847	120.641470533	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
848	120.642249181	198.51.100.1	203.0.113.2	SSHv2	210 Client: Encrypted packet (len=144)
849	120.643093633	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
850	120.644535521	198.51.100.1	203.0.113.2	SSHv2	962 Client: Encrypted packet (len=896)
851	120.645112156	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
852	120.645509081	198.51.100.1	203.0.113.2	SSHv2	146 Client: Encrypted packet (len=80)
853	120.646021212	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
854	120.647066576	198.51.100.1	203.0.113.2	SSHv2	194 Client: Encrypted packet (len=128)
855	120.647558407	203.0.113.2	198.51.100.1	SSHv2	162 Server: Encrypted packet (len=96)
856	120.648663374	198.51.100.1	203.0.113.2	SSHv2	210 Client: Encrypted packet (len=144)
857	120.649144904	203.0.113.2	198.51.100.1	SSHv2	146 Server: Encrypted packet (len=80)
858	120.649930652	198.51.100.1	203.0.113.2	SSHv2	3922 Client: Encrypted packet (len=3856)
859	120.650382380	203.0.113.2	198.51.100.1	TCP	66 22 → 36640 [ACK] Seq=4091 Ack=7857 Win=64128 Len=0 TSval=428358421 TSecr=687804198
860	120.650438983	198.51.100.1	203.0.113.2	SSHv2	146 Client: Encrypted packet (len=80)

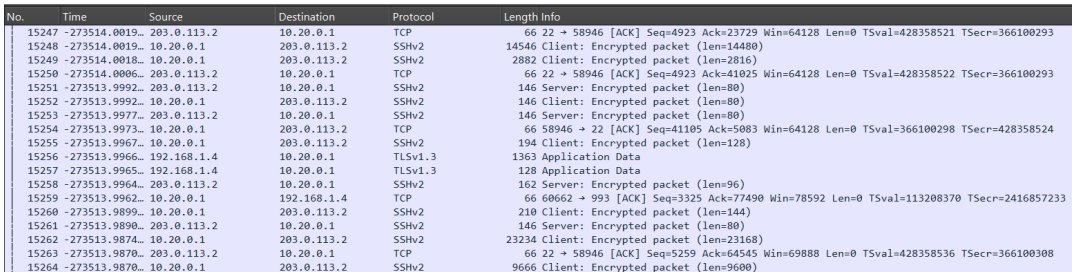
(a) List of Packets

```
▾ Frame 837: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on inte
    Section number: 1
  ▸ Interface id: 0 (eth5)
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 17, 2024 15:02:02.017173090 W. Europe Standard Time
    UTC Arrival Time: Nov 17, 2024 14:02:02.017173090 UTC
    Epoch Arrival Time: 1731852122.017173090
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000296918 seconds]
    [Time delta from previous displayed frame: 0.000296918 seconds]
    [Time since reference or first frame: 120.624536994 seconds]
    Frame Number: 837
    Frame Length: 130 bytes (1040 bits)
    Capture Length: 130 bytes (1040 bits)
```

(b) Packet Details

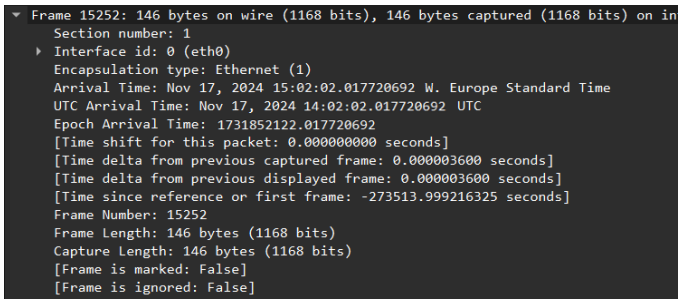
To determine which devices have been compromised by the malware, an effective approach involves analyzing traffic captures from routers within the network configuration. In this simulation, the internal network of the company is divided in three sub networks by three routers. The analysis focuses on the LAN router, operating within the IP range *10.20.0.0/24*. With prior knowledge of the traffic type to investigate, the captured router traffic can be analyzed efficiently. For example, it was observed that packets were sent from *10.20.0.1* to *203.0.113.2*, with their timestamps aligning with the frame in the Figure 9.1b. The same applies to other IP addresses, such as *10.20.0.3*. These findings confirm that one of the compromised devices is associated with the IP address *10.20.0.1*.

Figure 9.2: Wireshark Capture on the Firewall



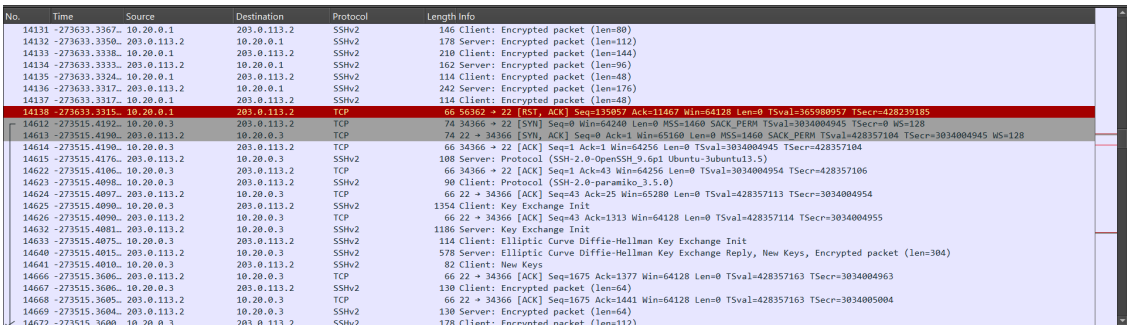
No.	Time	Source	Destination	Protocol	Length Info
15247	-273514.0019…	203.0.113.2	10.20.0.1	TCP	66 22 → 58946 [ACK] Seq=4923 Ack=23729 Win=64128 Len=0 TSval=428358521 TSecr=366100293
15248	-273514.0019…	10.20.0.1	203.0.113.2	SSHv2	14546 Client: Encrypted packet (len=14480)
15249	-273514.0018…	10.20.0.1	203.0.113.2	SSHv2	2882 Client: Encrypted packet (len=2816)
15250	-273514.0006…	203.0.113.2	10.20.0.1	TCP	66 22 → 58946 [ACK] Seq=4923 Ack=41025 Win=64128 Len=0 TSval=428358522 TSecr=366100293
15251	-273513.9992…	203.0.113.2	10.20.0.1	SSHv2	146 Server: Encrypted packet (len=80)
15252	-273513.9992…	10.20.0.1	203.0.113.2	SSHv2	146 Client: Encrypted packet (len=80)
15253	-273513.9977…	203.0.113.2	10.20.0.1	SSHv2	146 Server: Encrypted packet (len=80)
15254	-273513.9973…	10.20.0.1	203.0.113.2	TCP	66 58946 → 22 [ACK] Seq=41105 Ack=5083 Win=64128 Len=0 TSval=366100298 TSecr=428358524
15255	-273513.9967…	10.20.0.1	203.0.113.2	SSHv2	194 Client: Encrypted packet (len=128)
15256	-273513.9966…	192.168.1.4	10.20.0.1	TLSv1.3	1363 Application Data
15257	-273513.9965…	192.168.1.4	10.20.0.1	TLSv1.3	128 Application Data
15258	-273513.9964…	203.0.113.2	10.20.0.1	SSHv2	162 Server: Encrypted packet (len=96)
15259	-273513.9962…	10.20.0.1	192.168.1.4	TCP	66 60662 → 993 [ACK] Seq=3325 Ack=77490 Win=78592 Len=0 TSval=113208370 TSecr=2416857233
15260	-273513.9899…	10.20.0.1	203.0.113.2	SSHv2	210 Client: Encrypted packet (len=144)
15261	-273513.9890…	203.0.113.2	10.20.0.1	SSHv2	146 Server: Encrypted packet (len=80)
15262	-273513.9874…	10.20.0.1	203.0.113.2	SSHv2	23234 Client: Encrypted packet (len=23168)
15263	-273513.9870…	203.0.113.2	10.20.0.1	TCP	66 22 → 58946 [ACK] Seq=5259 Ack=64545 Win=69888 Len=0 TSval=428358536 TSecr=366100308
15264	-273513.9870…	10.20.0.1	203.0.113.2	SSHv2	9666 Client: Encrypted packet (len=9600)

(a) List of Packets



```
▾ Frame 15252: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on in
    Section number: 1
  ▸ Interface id: 0 (eth0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 17, 2024 15:02:02.017720692 W. Europe Standard Time
    UTC Arrival Time: Nov 17, 2024 14:02:02.017720692 UTC
    Epoch Arrival Time: 1731852122.017720692
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000003600 seconds]
    [Time delta from previous displayed frame: 0.000003600 seconds]
    [Time since reference or first frame: -273513.999216325 seconds]
    Frame Number: 15252
    Frame Length: 146 bytes (1168 bits)
    Capture Length: 146 bytes (1168 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
```

(b) Packet Details



No.	Time	Source	Destination	Protocol	Length Info
14131	-273633.3367…	10.20.0.1	203.0.113.2	SSHv2	146 Client: Encrypted packet (len=80)
14132	-273633.3350…	203.0.113.2	10.20.0.1	SSHv2	178 Server: Encrypted packet (len=112)
14133	-273633.3338…	10.20.0.1	203.0.113.2	SSHv2	210 Client: Encrypted packet (len=144)
14134	-273633.3333…	203.0.113.2	10.20.0.1	SSHv2	162 Server: Encrypted packet (len=96)
14135	-273633.3324…	10.20.0.1	203.0.113.2	SSHv2	114 Client: Encrypted packet (len=48)
14136	-273633.3317…	203.0.113.2	10.20.0.1	SSHv2	242 Server: Encrypted packet (len=176)
14137	-273633.3317…	10.20.0.1	203.0.113.2	SSHv2	114 Client: Encrypted packet (len=48)
14138	-273633.3315…	10.20.0.1	203.0.113.2	TCP	66 56362 → 22 [RST, ACK] Seq=135057 Ack=11467 Win=64128 Len=0 TSval=365980957 TSecr=428239185
14612	-273515.4192…	10.20.0.3	203.0.113.2	TCP	74 34366 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3034004945 TSecr=0 WS=128
14613	-273515.4190…	203.0.113.2	10.20.0.3	TCP	74 22 → 34366 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=428357104 TSecr=3034004945 WS=128
14614	-273515.4190…	10.20.0.3	203.0.113.2	TCP	66 34366 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3034004945 TSecr=428357104
14615	-273515.4176…	203.0.113.2	10.20.0.3	SSHv2	108 Server: Protocol (SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5)
14622	-273515.4106…	10.20.0.3	203.0.113.2	TCP	66 34366 → 22 [ACK] Seq=1 Ack=43 Win=64256 Len=0 TSval=3034004954 TSecr=428357106
14623	-273515.4098…	10.20.0.3	203.0.113.2	SSHv2	90 Client: Protocol (SSH-2.0-paramiko_3.5.0)
14624	-273515.4097…	203.0.113.2	10.20.0.3	TCP	66 22 → 34366 [ACK] Seq=43 Ack=25 Win=65280 Len=0 TSval=428357113 TSecr=3034004954
14625	-273515.4090…	10.20.0.3	203.0.113.2	SSHv2	1354 Client: Key Exchange Init
14626	-273515.4090…	203.0.113.2	10.20.0.3	TCP	66 22 → 34366 [ACK] Seq=43 Ack=1313 Win=64128 Len=0 TSval=428357114 TSecr=3034004955
14632	-273515.4081…	203.0.113.2	10.20.0.3	SSHv2	1186 Server: Key Exchange Init
14633	-273515.4075…	10.20.0.3	203.0.113.2	SSHv2	114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
14640	-273515.4015…	203.0.113.2	10.20.0.3	SSHv2	578 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=304)
14641	-273515.4010…	10.20.0.3	203.0.113.2	SSHv2	82 Client: New Keys
14666	-273515.3606…	203.0.113.2	10.20.0.3	TCP	66 22 → 34366 [ACK] Seq=1675 Ack=1377 Win=64128 Len=0 TSval=428357163 TSecr=3034004963
14667	-273515.3606…	10.20.0.3	203.0.113.2	SSHv2	130 Client: Encrypted packet (len=64)
14668	-273515.3605…	203.0.113.2	10.20.0.3	TCP	66 22 → 34366 [ACK] Seq=1675 Ack=1441 Win=64128 Len=0 TSval=428357163 TSecr=3034005004
14669	-273515.3604…	203.0.113.2	10.20.0.3	SSHv2	130 Server: Encrypted packet (len=64)

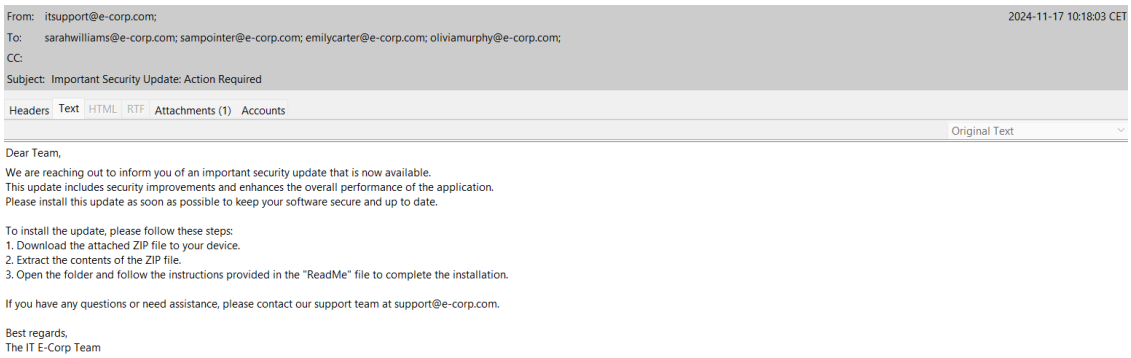
(c) Additional Packets

The next step involves investigating how the malware infiltrated the device. This is achieved by analyzing the compromised computer using forensic tools like Autopsy. The initial investigation focuses on identifying the origin of the malware, often through downloads or received emails. In this case, a review of the download folder revealed no unusual files, only some PDF documents downloaded previously, as shown in the Figure 9.3. The attention then shifted to emails received by the user. Browsing through the emails, one suspicious message stood out: an email presumably sent by the IT team to multiple recipients on *November 17th at 10:18 CET* (UTC+1), urging them to install immediately an attached update for security reasons, as shown in the Figure 9.4. Logs from the email server (Listing 9.1) then revealed that the sender of this email authenticated with the username *elliotalderson* and the IP address *203.0.113.2*, raising further suspicion.

Figure 9.3: Downloads with Firefox

Source Name	S	C	O	URL	Path	Date Accessed	Program Name	Domain
places.sqlite			12	https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resou..	home/sarahwilliams/downloads/dummy.pdf	2024-11-01 11:37:46 CET	FireFox Analyzer	w3.org
places.sqlite			12	https://www.irbnet.de/daten/iconda/CIB19103.pdf	home/sarahwilliams/downloads/CIB19103.pdf	2024-11-02 11:37:46 CET	FireFox Analyzer	irbnet.de
places.sqlite			8	https://cordis.europa.eu/docs/results/234127/final1-sta..	home/sarahwilliams/downloads/final1-stadium-final-re.	2024-11-01 13:17:06 CET	FireFox Analyzer	europa.eu
places.sqlite			8	https://www.fia.com/sites/default/files/fia_2024_formul..	home/sarahwilliams/downloads/fia_2024_formula_1_tec	2024-11-03 12:27:26 CET	FireFox Analyzer	fia.com
places.sqlite			9	https://www.imperva.com/resources/whitepapers/More.	home/sarahwilliams/downloads/More-Lessons-Learned.	2024-10-29 05:51:52 CET	FireFox Analyzer	imperva.com

Figure 9.4: Suspicious Email

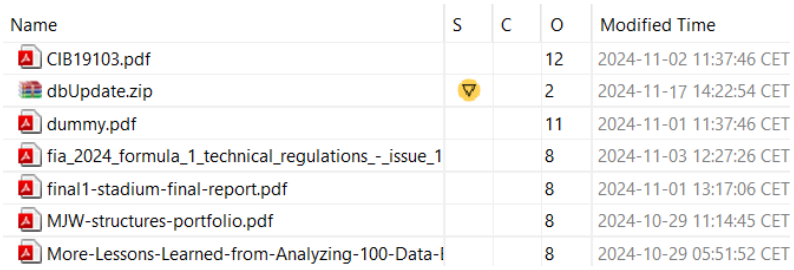


Listing 9.1: Email Server Logs

```
1  Nov 17 09:18:03 e-corp postfix/submission/smtpd[5471]: connect from unknown[203.0.113.2]
2  Nov 17 09:18:03 e-corp postfix/submission/smtpd[5471]: A7C4E9F2B3D:
       client=unknown[203.0.113.2], sasl_method=PLAIN, sasl_username=elliotalderson
3  Nov 17 09:18:03 e-corp postfix/cleanup[5475]: A7C4E9F2B3D:
       message-id=<Zzm0y-scgi-AySb0@74f38a131f97>
4  Nov 17 09:18:03 e-corp postfix/qmgr[5424]: A7C4E9F2B3D: from=<itsupport@e-corp.com>,
       size=7943, nrcpt=4 (queue active)
5  Nov 17 09:18:03 e-corp postfix/submission/smtpd[5471]: disconnect from unknown[203.0.113.2]
       ehlo=2 starttls=1 auth=1 mail=1 rcpt=4 data=1 quit=1 commands=11
6  Nov 17 09:18:03 e-corp postfix/local[5476]: A7C4E9F2B3D: to=<emilycarter@e-corp.com>,
       relay=local, delay=0.15, delays=0.09/0.02/0/0.04, dsn=2.0.0, status=sent (delivered to
       mailbox)
7  Nov 17 09:18:03 e-corp postfix/local[5477]: A7C4E9F2B3D: to=<oliviamurphy@e-corp.com>,
       relay=local, delay=0.18, delays=0.09/0.04/0/0.05, dsn=2.0.0, status=sent (delivered to
       mailbox)
8  Nov 17 09:18:03 e-corp postfix/local[5476]: A7C4E9F2B3D: to=<sampointer@e-corp.com>,
       relay=local, delay=0.19, delays=0.09/0.06/0/0.04, dsn=2.0.0, status=sent (delivered to
       mailbox)
9  Nov 17 09:18:03 e-corp postfix/local[5478]: A7C4E9F2B3D: to=<sarahwilliams@e-corp.com>,
       relay=local, delay=0.2, delays=0.09/0.07/0/0.04, dsn=2.0.0, status=sent (delivered to mailbox)
10 Nov 17 09:18:03 e-corp postfix/qmgr[5424]: A7C4E9F2B3D: removed
```

The investigation proceeded to determine whether the user had downloaded and installed the update. Using Autopsy's Keyword Search module to look for *dbUpdate*, it was confirmed that the zip file was present in the downloads folder, but there was no evidence of the extracted folder. From the Figure 9.5, it is possible to notice that the victim saved the attachment on *November 17th at 14:22 CET* (UTC+1). This left two possibilities: either the file had not been extracted yet, or it had been extracted, installed, and subsequently deleted. By examining the bash command history, further evidence was found indicating that the folder had indeed been extracted, the fake update installed, and the folder deleted afterward (see Listing 9.2). This confirmed that the user had followed the email instructions and that the malware should be inside the computer.

Figure 9.5: dbUpdate Inside Download Folder



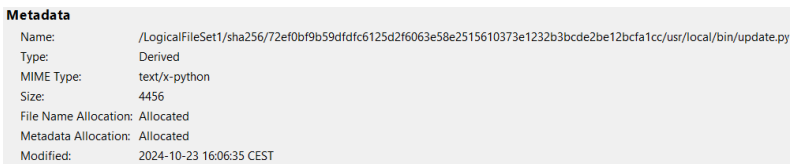
Name	S	C	O	Modified Time
CIB19103.pdf			12	2024-11-02 11:37:46 CET
dbUpdate.zip			2	2024-11-17 14:22:54 CET
dummy.pdf			11	2024-11-01 11:37:46 CET
fia_2024_formula_1_technical_regulations_-_issue_1			8	2024-11-03 12:27:26 CET
final1-stadium-final-report.pdf			8	2024-11-01 13:17:06 CET
MJW-structures-portfolio.pdf			8	2024-10-29 11:14:45 CET
More-Lessons-Learned-from-Analyzing-100-Data-I			8	2024-10-29 05:51:52 CET

Listing 9.2: Command Bash History

```
cd downloads
ls
clear
unzip dbUpdate.zip
clear
ls dbUpdate
cd dbUpdate
clear
cat README.txt
clear
python3 installer.py
clear
cd ..
rm -r dbUpdate
clear
```

To understand the behavior of the malware, the email attachment was extracted (Figure 8.6), and its content analyzed. The attachment contained a zip file (*dbUpdate.zip*), which, when unzipped, revealed several files: an installer script (*installer.py*), a readme file, a *requirements.txt*, and an update folder containing the malicious update script. The *installer.py* script served as the entry point of the malware. Upon execution, the script determined its location, installed required packages listed in the *requirements.txt* file, and ensured that the *update.py* script was moved to a specific directory. It then added *update.py* to the crontab scheduler for periodic execution across all user accounts that are in the computer, ensuring persistence even if the original folder was deleted. Additionally, the script activated the crontab service. To confirm the malware presence, the crontab configuration was inspected (Figure 9.7), revealing the scheduled execution of the *update.py* script at the specified path (*/usr/local/bin/update.py*) where *installer.py* moved it (Figure 9.6). The purpose of the malicious script was to send the contents of targeted folders, specifically those within */home/username/*, to a remote server with *203.0.113.2* as IP address via SSH over port 22. By utilizing the paramiko library, the script established remote connections to transfer the compromised data securely, allowing the attacker to create remotely, from the victim's computer, the destination folders for each users with sub folders inside of it for each time the script was run.

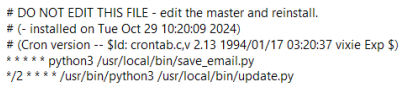
Figure 9.6: Malicious Script Location



Metadata

Name:	/LogicalFileSet1/sha256/72ef0bf9b59dfdfc6125d2f6063e58e2515610373e1232b3bcde2be12bcfa1cc/usr/local/bin/update.py
Type:	Derived
MIME Type:	text/x-python
Size:	4456
File Name Allocation:	Allocated
Metadata Allocation:	Allocated
Modified:	2024-10-23 16:06:35 CEST

Figure 9.7: Crontab Configuration



9.3.1 Result of the Investigation

The analysis reveals that the malware was distributed by an attacker, identified as Elliot Alderson, to four employees under the guise of a critical security update (Figure 9.4). This email, sent on *November 17th at 09:18 UTC*, included an attachment named *dbUpdate.zip* that supposedly contained the update. By examining network traffic captured by the routers within the system architecture, it was determined that three employees installed the update (Figure 9.2c). The procedures of the investigation focused on the case of Sarah Williams, whose device had the IP address *10.20.0.1* (Figure 9.2b). In Sarah's case, the malicious file *dbUpdate.zip* was saved to the computer on *November 17 at 13:22 UTC* (Figure 9.5), after which the update was installed. Following the installation, Sarah deleted the folder, unaware that malicious code remained active on the system (Listing 9.2). Shortly thereafter, the malware began its operations, which were observable through network traffic analysis. Using Wireshark, a series of encrypted packets transmitted via SSH were detected, raising suspicion and confirming the activity of the malware (Figure 9.2a).

Chapter 10

Future Works

This thesis examines various types of cyberattacks targeting small and medium-sized businesses by reconstructing company infrastructures in controlled environments using Docker and simulating different attack scenarios. The objective was to execute these attacks and perform detailed analyses to understand their mechanisms. The analysis primarily relied on examining architectural components through tools such as Autopsy. Depending on the type of attack, additional data, including system logs from email and database servers and network traffic captures at strategic locations like firewalls and routers within the simulated network, were analyzed. This approach provided a comprehensive understanding of the attack scenarios and informed the development of potential preventive measures to mitigate future risks.

The research focused on four main types of attacks: data breaches, phishing, ransomware, and malware. Since cyberattacks can be executed using various methodologies, this study opens opportunities for further exploration by investigating alternative methods of execution. The simulated attacks in this research heavily employed social engineering techniques, particularly through the use of malicious emails to deceive victims. Future studies could extend this work by employing alternative strategies, such as exploiting system vulnerabilities to gain unauthorized access or distributing malware via links to malicious websites that prompt the download of files containing harmful code disguised as legitimate.

Simulating a variety of attack types with diverse methodologies not only enhances understanding of these threats but also generates valuable insights during digital forensic analysis. These findings contribute to the development of strategies for preventing such attacks and improving readiness to respond effectively when they occur. Beyond focusing on attack types, the study also highlights the potential benefits of recreating company infrastructures with varying device configurations or different operating system versions. While Linux was primarily used in this research, incorporating a mix of systems, such as computers running Windows alongside mobile devices operating on Android or iOS, could offer additional perspectives. For instance, integrating mobile devices into the simulated environments allows the recreation of attacks exploiting human error to obtain credentials or sensitive data, further enriching the scope of analysis and understanding of potential vulnerabilities.

An enhancement to the simulated company infrastructure that could be beneficial for digital forensic investigations involves the implementation of both public and private certificates for client-server authentication. This approach utilizes private certificates for internal authentication while employing public certificates for external authentication. As a result, the Certificate Authority (CA) is divided into public and private entities. The CA, as a trusted institution responsible for issuing digital certificates, plays a critical role in securing online interactions by cryptographically linking an identity to a public key. This configuration enables secure authentication between devices within the network and external servers through the firewall while incorporating a Man-in-the-Middle (MITM) mechanism. In this scenario, the device initiates a TLS session under the assumption that it is communicating directly with the intended server. However, the firewall intercepts this traffic and responds with a certificate that, instead of being the server's public certificate, is a private certificate issued by the internal Certificate Authority (CA). If the device recognizes and trusts the internal CA, it accepts the connection. The firewall then establishes a separate TLS session with the actual external server, this time using the server's legitimate public

certificate. Once this process is completed, a secure TLS session is formed between the firewall and the external server. Since the firewall effectively terminates and re-encrypts traffic using a certificate under its control, it gains access to the plaintext data being transmitted. This capability allows the firewall to log, inspect, or forward decrypted traffic as necessary. By capturing packets on the firewall, access to the private key-generated along with the internal certificate-enables decryption and in-depth analysis of the network traffic using tools such as Wireshark. This feature improves the ability to analyze encrypted communications, providing valuable information for forensic investigations while enabling more effective monitoring of user activity within the corporate network.

Bibliography

[1] IBM, "What is virtualization?", https://www.ibm.com/topics/virtualization

[2] Keith Ward, "The top 5 enterprise type 1 hypervisors you must know", April 2021, https://www.actualtechmedia.com/io/top-5-enterprise-type-1-hypervisors/

[3] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues", April 2010, DOI 10.1109/ICCNT.2010.49, https://ieeexplore.ieee.org/document/5474503

[4] Amazon, "What is virtualization?", https://aws.amazon.com/what-is/virtualization/

[5] Lewis Andrews, "Enhancing security through virtualization technology", September 2023, https://www.franklinfitch.com/uk/resources/blog/enhancing-security-through-virtualization-technology/

[6] Priyanka Kulkarni Joshi, "Digital forensics 2.0: Innovations in virtual environment and emerging technologies", April 2023, https://www.eccouncil.org/cybersecurity-exchange/computer-forensics/digital-foresics-innovations-emerging-technologies/

[7] Autopsy Documentation, "Autopsy user documentation", 2024, https://sleuthkit.org/autopsy/docs/user-docs/4.21.0/

[8] Malware News, "Introduction to memory forensics with volatility 3", 2024, https://malware.news/t/introduction-to-memory-forensics-with-volatility-3/79168

[9] Volatility Documentation, "Volatility 3 basics", 2024, https://volatility3.readthedocs.io/en/latest/basics.html

[10] Neil Fox, "How to use volatility for memory forensics and analysis", 2023, https://www.varonis.com/blog/how-to-use-volatility

[11] Kody Kinzie, "How to use wireshark", August 2022, https://www.varonis.com/blog/how-to-use-wireshark

[12] Edward Kost, "What is wireshark? the free network sniffing tool", 2024, https://www.upguard.com/blog/what-is-wireshark

[13] My F5, "K50557518: Decrypt ssl traffic with the sslkeylogfile environment variable on firefox or google chrome using wireshark", February 2023, https://my.f5.com/manage/s/article/K50557518

[14] Parker Benitez, "Wireshark network and malware analysis", September 2023, https://medium.com/@parkerbenitez/wireshark-traffic-and-malware-analysis-2a5da9b5a610

[15] Alexander S. Gillis, "What is security information and event management (siem)?", June 2024, https://www.techtarget.com/searchsecurity/definition/security-information-and-event-management-SIEM

[16] Fortinet, "What is siem?", 2024, https://www.fortinet.com/resources/cyberglossary/what-is-siem

[17] Duja Consulting, "Benefits of conducting a proactive forensic investigation with case studies", June 2023, https://www.linkedin.com/pulse/benefits-conducting-proactive-forensic-investigation-case/

[18] Optiv, "What is data forensics / forensic data analysis (fda)?", https://www.linkedin.com/pulse/benefits-conducting-proactive-forensic-investigation-case/

[19] Optiv, "What is database forensics?", May 2022, https://www.salvationdata.com/knowledge/what-is-database-forensics/

[20] Geeksforgeeks, "Techniques of cyber forensics", Apr 2024, https://www.geeksforgeeks.org/techniques-of-cyber-forensics/

[21] BlueBoyant, "Understanding digital forensics process techniques and tools", https://www.bluevoyant.com/knowledge-center/understanding-digital-forensics-process-techniques-and-tools

[22] Soufiane Tahiri, "Digital forensics models", January 2016, https://www.infosecinstitute.com/resources/digital-forensics/digital-forensics-models/

[23] Priya S. Patil and A. S. Kapse, "Survey on different phases of digital forensics investigation models", March 2015, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=bd034b87414c58fc731c193fdd37f59b3db883fa

[24] Sudesh Rani, "Digital forensic models: A comparative analysis", June 2018, https://www.ijmra.us/project%20doc/2018/IJMIE_JUNE2018/IJMRA-14015.pdf

[25] Dominique Brezinski and Tom Killalea, "Guidelines for evidence collection and archiving", February 2002, DOI 10.17487/RFC3227, https://www.rfc-editor.org/rfc/pdfrfc/rfc3227.txt.pdf

[26] Venansius Baryamureeba and Florence Tushabe, "The enhanced digital investigation process model", May 2004, https://dfrws.org/wp-content/uploads/2019/06/2004_USA_paper-the_enhanced_digital_investigation_process_model.pdf

[27] Ankit Agarwal and Megha Gupta and Saurabh Gupta and Subhash Chandra Gupta, "Systematic digital forensic investigation model", January 2011, https://www.researchgate.net/publication/228410430_Systematic_Digital_Forensic_Investigation_Model

[28] James R. Lyle and Barbara Guttman and John M. Butler and Kelly Sauerwein and Christina Reed and Corrine E. Lloyd, "Digital investigation techniques: A nist scientific foundation review", November 2022, DOI 10.6028/NIST.IR.8354, https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8354.pdf

[29] Shmuel Gihon, "Ransomware trends 2023 report", April 2024, https://cyberint.com/blog/research/ransomware-trends-and-statistics-2023-report/

[30] Shmuel Gihon, "Ransomware groups report 2024 - q3", October 2024, https://cyberint.com/blog/research/ransomware-trends-2024-report/

[31] Nicole Thompson, "How the toronto library is recovering from a cyberattack, one book at a time", March 2024, https://www.canadianunderwriter.ca/insurance/how-the-toronto-library-is-recovering-from-a-cyberattack-one-book-at-a-time-1004243402/

[32] Luke Irwin, "Criminal hackers leak email addresses of 220 million twitter users", January 2023, https://www.itgovernance.co.uk/blog/criminal-hackers-leak-email-addresses-of-220-million-twitter-users

[33] Twingate Team, "What happened in the community health systems data breach?", May 2024, https://www.twingate.com/blog/tips/community-health-systems-data-breach

[34] Anne Campbell, "Report: 2023 spear-phishing trends", May 2023, https://blog.barracuda.com/2023/05/24/2023-spear-phishing-trends

[35] Trustpair, "4 examples of spear phishing attacks", https://trustpair.com/blog/4-examples-of-spear-phishing-attacks/

[36] Cyberhaven, "Top ip theft statistics and stories in 2023", January 2024, https://www.cyberhaven.com/guides/top-ip-theft-statistics

[37] Yana Storchak, "Top 10 best-known cybersecurity incidents and what to learn from them", May 2024, https://www.syteca.com/en/blog/top-10-best-known-cybersecurity-incidents-and-what-to-learn-from-them

[38] Umar Shakir, "Tesla points to insider wrongdoing as cause of massive employee data leak", August 2023, https://www.theverge.com/2023/8/21/23839940/tesla-data-leak-inside-job-handelsblatt

[39] Kasperky ICS CERT, "Apt attacks on industrial companies in 2020", March 2021, https://ics-cert.kaspersky.com/publications/reports/2021/03/29/apt-attacks-on-industrial-companies-in-2020/

[40] Hackerone, "Advanced persistent threats: Attack stages, examples, and mitigation", https://www.hackerone.com/knowledge-center/advanced-persistent-threats-attack-stages-examples-and-mitigation

[41] Baivab Kumar Jena, "Solarwinds attack and all the details you need to know about it", August 2024, https://www.simplilearn.com/tutorials/cryptography-tutorial/all-about-solarwinds-attack

[42] dockerdocs, "Why use compose?", https://docs.docker.com/compose/intro/features-uses/
[43] dockerdocs, "Network drivers", https://docs.docker.com/engine/network/drivers/
[44] dockerdocs, "Docker compose", https://docs.docker.com/reference/cli/docker/compose/
[45] CheckPoint, "Phishing detection techniques", https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-phishing/phishing-detection-techniques/
[46] asee, "Phishing attacks: How to recognize and protect your organization from phishing scams", 2023, https://cybersecurity.asee.io/blog/phishing-attacks-how-to-recognize-phishing-scams/
[47] Philip Robinson - Lepide, "How to detect ransomware: Common detection techniques", 2024, https://www.lepide.com/blog/how-to-detect-ransomware/
[48] PowerAdmin, "Use honeypot folders to detect ransomware", https://www.poweradmin.com/products/file-sight/resources/setup-guide-for-ransomware-protection/ransomware-detection-with-honeypots/
[49] Kurt Baker - CrowdStrike, "10 malware detection techniques", 2023, https://www.crowdstrike.com/en-us/cybersecurity-101/malware/malware-detection/