POLITECNICO DI TORINO

Master Degree Course in Cybersecurity

Master Degree Thesis

Real-Time Automated Forensic Evidence Collection in Critical Systems

Leveraging Advanced Network Monitoring Tools for Enhanced Cybersecurity Incident Response

Supervisor
prof. Andrea Atzeni

Candidate
Pierfrancesco Elia

Academic Year 2024-2025

Alla mia famiglia,
ai miei amici,
alle persone che mi vogliono bene.

Summary

This thesis addresses an important challenge in modern cybersecurity: the vast gap between threat detection and forensic examination that allows attackers to steal sensitive data or erase evidence within minutes but traditional forensic response measures take hours or even days to initiate. The average data breach in the world costs $4.88 million USD, as per the 2024 Cost of a Data Breach Report from IBM, and companies that respond more quickly save around $2.22 million compared with slower-responding peers, highlighting the financial imperative behind automated forensic readiness. The research shows how the open-source network monitoring platform Zabbix can be innovatively repurposed to enable in real-time the gathering of forensic evidence with legal admissibility standards in mind, tailored in particular to the needs of environments with scarce resources, air-gapped networks, industrial control networks, and legacy infrastructures in which standard SIEM/SOAR deployments become infeasible.

Existing enterprise forensic solutions pose key shortcomings that make this research trajectory imperative. Vendor SIEM systems such as Splunk Enterprise Security (high renewal cost) and IBM QRadar (high deployment fees) are far too costly for 85% of organisations, while being highly dependent on cloud connectivity as well as requiring specialised staff. Current solutions such as Elastic SIEM rely on internet connectivity that is unsuitable for air-gapped industrial networks, while established forensics software such as EnCase is based on post-incident reactive models that do not capture volatile evidence in real-time active attacks. This leaves a market need where 70% of organisations do not have proper forensic automation in their toolkit, especially in industrial control systems as well as resource-limited settings where common SIEM/SOAR installations become technically and financially impracticable.

The basic breakthrough is in modular design comprising four pieces, which translates the capabilities of Zabbix for infrastructure monitoring into an inclusive platform of forensic automation. The Detection Module uses the underlying trigger system of Zabbix to identify security-related patterns with the aid of logical expressions, temporal functions, and statistical correlations, focused on indicators of compromise instead of typical performance measures. Tailored triggers utilize correlation logic that blends multiple telemetry sources, such as anomalies in network traffic, unusual patterns of process execution, and occurrences of failed authentication, while accomplishing sophisticated mitigation of false positives through contextual filtering, temporal examination, as well as hierarchical frameworks of dependencies.

The Response Module manages automated responses through the action framework of Zabbix, enabling the execution of scripts triggered by security-related events while maintaining contextual information through macro expansion and runtime parameters. The module acts as an intelligent dispatcher that initiates evidence-gathering operations independently with escalation plans and cooldown times in order to prevent redundant runs. The Acquisition Module is composed of special-purpose scripts written specifically for the extraction of volatile and semi-volatile evidence such as memory dumps, packet captures, process trees, system logs, and network connections, with each script running autonomously in order to prevent data contamination while outputting structured metadata and verifying cryptographic integrity.

The Preservation Module enacts extensive procedures for chain-of-custody using SHA-256 hashing, time-stamping synchronized with NTP, and storage mechanisms that are tamper-evident, thereby ensuring legal admissibility. This module preserves thorough audit trails that document

collection methods, timestamps, triggering conditions, and script executions, in addition to facilitating encrypted transmission to safe remote repositories. The architectural framework prioritizes the separation of concerns, permitting individual components to be independently versioned, tested, and deployed while ensuring cohesive integration through the native orchestration capabilities of Zabbix.

Empirical verification performed in controlled lab experiments identifies exemplary performance characteristics that are valid for production environments. Two in-depth case studies verify the effectiveness of the framework in diverse threat scenarios. The first case study emulates data exfiltration in measuring the unauthorized movement of 30 GB of company information through SFTP, accurately detecting prolonged outbound traffic anomalies in excess of 10 Mbps thresholds and collecting forensically significant artifacts that provides a detailed system state snapshot. The second case study analyzes unauthorized service provisioning with sophisticated backdoor detection using Zabbix Low-Level Discovery in monitoring industrial network services, accurately detecting services deployed in Python that conceal themselves as genuine diagnostic interfaces, thus collecting unique evidence regarding the incident.

Forensic integrity verification confirms all evidence gathered remains legally admissible with dual-layer SHA- 256 hashing, full metadata documentation, and NTP time sync. Individual artifacts are immediately crypto-verified at the point of creation, while full evidence stores undergo holistic verification that covers all materials included. Chain-of-custody protocols involve meticulous recordation of the means of collection, script versions, system configurations, and operator context, with encrypted transmission over protected channels and tamper-evident storage with write-once, read-many access protection.

Rigorous testing proves quantitative superiority over available methods in significant performance measures. The system realizes important cost savings compared to commercial forensic automation solutions while achieving high detection accuracy. Response time optimization realizes 30-second initiation of evidence collection compared to 2-4 hour delays in legacy forensic processes. The solution remedies essential deployment scenarios where commercial solutions fall short: air-gapped networks (100% offline operability), legacy industrial applications (no-agent requirements), and small-medium businesses (zero licensing fees). Forensic integrity assurance by dual-layer SHA-256 hashing and detailed chain-of-custody documentation secures legal admissibility standards identical to enterprise solutions at infinitesimally lower implementation costs, bringing automated forensic capabilities within reach of organizations previously priced out by commercial offerings.

The framework addresses critical needs across multiple sectors including industrial control systems where traditional security tools prove unsuitable due to real-time constraints or legacy system limitations, air- gapped networks requiring forensic capabilities without external connectivity, small-medium enterprises needing cost-effective alternatives to expensive commercial platforms, and regulatory compliance supporting GDPR, HIPAA, and PCI-DSS requirements through comprehensive audit trails and evidence preservation mechanisms. Future directions involve the integration with machine learning for better detection of anomalies, container forensics extending the capability into Docker or Kubernetes deployments, cloud evidence collecting modifying methodologies in support of AWS CloudTrail and Azure Activity Logs, and industrial protocol analysis offering custom modules for Modbus and DNP3 inspection of network traffic. Modular design allows for extensibility via standard input/output descriptions that facilitate the incorporation with various other monitoring platforms in addition to the Zabbix. This research effectively demonstrates that the established network monitoring infrastructures can be skillfully re-purposed as comprehensive forensic automation infrastructures with no requirements for special devices or significant additional costs. The proven framework empowers enterprises with practical, legally acceptable, and functionally efficient response systems that handle incidents, particularly in resource-constrained environments in which commercial solutions are frequently out of the question. The empirical verification through controlled experiments verifies technical possibility as well as forensic legitimacy and forms the foundation of wide-scale use in diverse application scenarios while driving the field of automated digital forensics through the novel application of available monitoring infrastructure.

Contents

Chapter 1

Introduction

Digital forensics is increasingly seen as a vital element within the discipline of modern-day cybersecurity, especially in high-risk scenarios where swift identification and response to intrusions are essential. Forensic practices are generally manual and follow the incident; however, the move toward automation systems and quickly performed analytical techniques offers exceptional potential to close the gap between detection and investigation. This dissertation explores the integration of monitoring systems into the process of digital forensics, with specific emphasis on the deployment of Zabbix as a leading open-source monitoring tool.

The conversation begins by considering situations where forensic preparedness is of critical need, focusing mostly on operational environments like industrial networks and air-gapped systems. Then the conversation spells out the main goal of the study: to clarify how a particular monitoring device, initially intended for purposes other than security, can be adapted to enable real-time and automated collection of forensic evidence.

Later studies follow a systematic approach. First, they review existing developments relevant to digital surveillance and forensics. Next, they present a conceptual evidence collection framework with Zabbix, culminating in the conducting of test cases in virtual environments, finally leading to an evaluation of the tool's advantages and disadvantages. The overall goal remains the significant enhancement of the burgeoning field of proactive forensics with an effective, flexible, and affordable measure, while simultaneously investigating how forensic readiness could be improved in resource-constrained environments.

1.1 Why Timing is Pivotal in Security Incidents

In the modern cybersecurity field, a consensus among specialists is that organizations can be expected to face a security breach; hence, the question is not whether a breach is to be expected, but when. This fact has been supported by ongoing data brought forth in yearly sector publications, such as IBM's Cost of a Data Breach Report [1], which in 2024 reported the global average cost of a breach at a record 4.88 million USD. Furthermore, the report highlights that not only do breaches occur more often, but also become progressively more aggressive as a result of longer detection and response times.

In this case, the two most critical steps are Time to Detect (TTD) and Time to Respond (TTR). They specify the duration from the compromise inception to its discovery and the time required to effectively respond to the scenario post-detection of the breach. Organizations that reduce TTD and TTR not only have a greater chance of containing attacks sooner but also of preserving useful evidence, maintaining business continuity, and reducing monetary and reputational damage. IBM's figures suggest that organizations with faster response times save around 2.22 million USD compared to organizations with more lengthy incident management processes.

Real-time incident detection is thus a fundamental aspect of contemporary cyber defense mechanisms. However, in most contexts, the response is still predominantly reactive and highly

dependent on human intervention. Analysts must manually inspect logs and alerts to discover anomalies, which could be the time the intruder has already exfiltrated sensitive information, turned off logging, or propagated laterally across systems. This lag is particularly dangerous in high-value or time-sensitive environments, such as medical, industrial control systems, or military installations, where attackers can cause real harm within minutes.

Furthermore, as cyber threats become more complex, the chance to spot and fix them before they cause damage is getting smaller. Advanced Persistent Threats (APTs), insider threats, and fileless malware operate quietly and quickly. As a result, even a short delay can lead to serious consequences, like data corruption, service outages, or legal penalties.

To address this, more organizations are turning to automation in their security processes to not only detect but also respond to threats instantly. This includes using automated tools to gather forensic data the moment an incident is suspected. Automating these responses can help capture important evidence, start isolation procedures, and support security teams with incident management before analysts begin their manual investigations. Therefore, this thesis concludes that timing is the key requirement for a safe and reliable response to cyberattacks.

The diagram in Figure 1.1 shows the links between the detection gap, response gap, and prevention gap. Together, these three areas emphasize how crucial timing is for each process, as it directly affects an organization's ability to limit damage and build resilience.

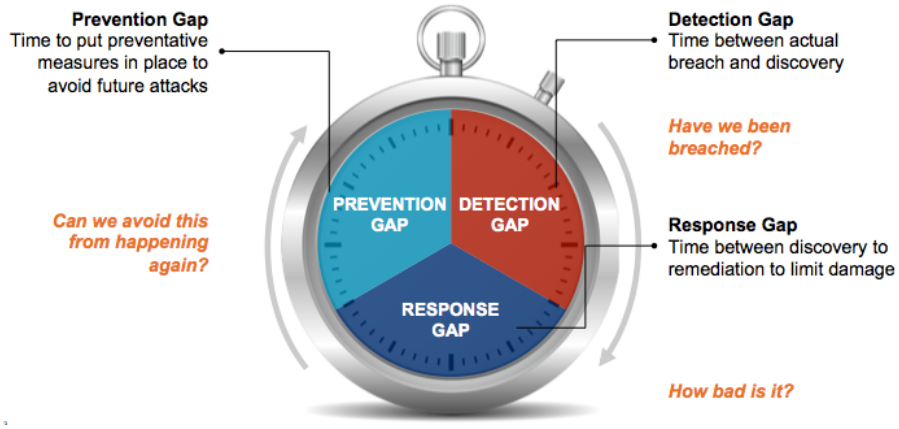

Figure 1.1. Detection, response, and prevention response gaps in incident management.

1.2 Case Studies of Real-World Security Failures

The importance of timely detection and response is described through documented cybersecurity breaches. Real-life incidents often show how minor delays or missed alarms can lead to large-scale incidents. This chapter section two of the most interesting cases: the 2024 CrowdStrike outage and the Target data breach of 2013, each highlighting examples of the consequences of a lack of timing and inadequate response strategy.

In July 2024, a flawed software update issued by the cybersecurity firm CrowdStrike [2] caused one of the most disastrous security incidents in recent years. The update spread in minutes to millions of Windows-based systems, triggering system crashes in hospitals, airports, government agencies, and industrial control networks. The problem was caused by an incompatibility related to Falcon's security agent, but was exacerbated by the absence of a real-time rollback process and the rushed release of the update without warning. The timing of this incident was crucial; a few minutes caused worldwide downtime. Manual intervention was necessary to recover several systems, and the presence of BitLocker encryption on organizational systems hindered forensic analysis and data recovery activities. The CrowdStrike incident demonstrates that a patch with good intentions can escalate into a huge failure in the absence of an automated monitoring or containment process.

A second example, while more historical, remains pertinent: the 2013 Target breach [3]. Threat actors had compromised the retailer's network through credentials stolen from a third-party vendor. Despite the presence of security controls, including malware detection capabilities, that picked up the malicious behavior, the alarms were either ignored or deprioritized by security staff. The attackers exfiltrated sensitive payment card information for over 40 million customers before the breach was discovered. Investigation showed that the attack could have been prevented much earlier if the warnings had been heeded in a timely fashion. In this case, it was the delay in response - not the failure to detect - that was the root cause of the breach's success. These examples demonstrate a shared principle: the identification of threat is not sufficient without prompt and concerted action. They also indicate that conventional security mechanisms are not always sufficient in the face of dynamic threat environments. If either entity had incorporated real-time automated forensic response capability into their surveillance systems, the exposure time could have been drastically shortened.

With the above limitations in mind, the current dissertation promotes the creation of methods that combine real-time response functions with holistic forensic mechanisms. The suggested methods should be able to respond to incidents in real-time, independently capture relevant information, and store such information for later analysis and possible prosecution purposes.

1.3 Accuracy and Integrity in Automated Forensics

Although timing is crucial in handling security incidents, the concept of speed must be approached carefully in the field of digital forensics. Rapid responses are essential to mitigate ongoing attacks alongside limiting damage, yet the forensic evidence gathered has to be trustworthy and admissible in legal processes. Hence, it is implied that evidence collection can not solely rely on speed, moreover, it must also ensure accuracy, integrity, and completeness.

In traditional forensic procedures, trained personnel manually handle evidence acquisition. These professionals follow set protocols to maintain the chain of custody. They use hashing algorithms to ensure data integrity and store the collected evidence in secure devices or distributed storage. While this process is very reliable, it is also slow and takes up a lot of resources.

In fast-paced environments, such as critical infrastructure, defense systems, or industrial networks, delays in evidence collection can lead to the loss of volatile data or give adversaries a chance to cover their tracks.

Automating processes presents both opportunities and risks. On one hand, automated systems can quickly respond to detect incidents, running predefined scripts to gather logs, memory dumps, or network traces without human input. However, this automation must not undermine the forensic validity.

The challenges of automatic evidence collection include ensuring that:

- collected data is not modified or overwritten during the process;
- appropriate hashing and time-stamping are applied at collection time;
- scope of collection is carefully defined to avoid irrelevant data;
- evidence is securely stored and access-controlled to maintain the chain of custody.

Evidence collection is a complex process, and one example is Microsoft's BitLocker [4], a full-disk encryption mechanism widely used among enterprises to secure company data stored inside work PC's of their employees. Automated forensic tools must be able to interact with encrypted systems without violating security policies or corrupting the content. This process requires precise timing: in fact, it is essential to acquire live data from RAM or disk images before the encryption mechanism seals the system, or while a trusted access is still active.

Additionally, automated collection must also account for context: what is considered relevant evidence varies depending on the type of incident. For example, in case of a brute-force attack

login attempt logs and authentication system responses are critical; in a data exfiltration scenario, network packet captures and process execution records may be more relevant.

Although the timing of security incidents is of the utmost significance, the concept of swiftness must be carefully considered by the field of digital forensics. Timely responses are important in reducing current threats alongside minimizing damage; however, the evidence recovered in such situations must be valid as well as acceptable in a court of law. Therefore, it can be justifiably stated that the evidence collection process cannot be based solely on swiftness; alongside it must emphasize accuracy, authenticity, as well as thoroughness.

1.4 Purposes and Goals

Given the growing prevalence of cyber threats and the urgency to institute responsive measures, the author has explored in this thesis the possibility of bringing network monitoring in line with automated processes of forensic data gathering. The hope here is not to replace traditional forensic methods but to create an additional system that can instantly react to anomalies as found to aid in preserving evidence through an automation system.

The key question that guides the focus of this inquiry is:

> How can a network monitoring tool like Zabbix be effectively used not just for the real-time detection of security incidents, but also for initiating and supporting automated forensic evidence gathering without compromising accuracy and legal admissibility?

In answering this question, the thesis provides a systematic approach that includes both a theoretical examination and an empirical investigation of the question under study.

The first step is to create an automation framework that responds directly to network events being monitored by Zabbix. This is a matter of identifying what kind of incidents should be handled automatically, what information needs to be gathered, and what tools to explore in conjunction with Zabbix to accomplish these things.

The secondary goal is to perform an in-depth evaluation of Zabbix's feasibility as a possible solution for the task at hand. While Zabbix is well known as an open-source monitoring system, its usability for forensic use is not adequately addressed in most common setups.

The purpose of this work is to present a practical, technically sound, and legally aware contribution to forensic readiness by exhibiting that proactive forensic techniques and monitoring enable quicker response rates while also protecting critical data.

Chapter 2

Background and State of the Art

Before implementing the recommended resolution, it's important to investigate the theoretical background in combination with the practical frameworks making up this research. The chapter includes an in-depth examination of key concepts dealing with digital forensics, incident response, as well as network monitoring, along with an in-depth investigation into the history surrounding automated forensic technology incorporation into current technology.

It covers an analytical comparison between traditional forensic tools and procedures against modern, real-time, and automated ones. Special attention goes to surveillance technology and its impact on security, exploring how common tools, such as SIEM systems, intrusion detection, and log management, enable forensic operations.

The final comments in the paper acknowledge Zabbix as a viable candidate with great potential for utilization in forensic environments. While Zabbix has mostly been utilized through infrastructure models, its modularity, combined with its scripting functionality, indicates potential for diverse utilization. This section presents the background context required for the discussion in the following chapter, explaining Zabbix's use in forensic environments while pointing out additional settings in which it can be used. The argument frames both the strengths and weaknesses tied to Zabbix, thus enabling deeper critical appraisal in the future.

2.1 Digital Forensics and Incident Response

Digital Forensics and Incident Response, or DFIR, combines two cybersecurity fields to streamline threat response while preserving legally and technically sound evidence against criminals.

National Institute of Standards and Technology defines digital forensics as a process that involves identifying, capturing, analyzing, and investigating data while maintaining data integrity and keeping at all times a rigorous chain of custody [5]. Digital forensics can handle a range of data sources such as media files, logs, memory dumps, and network communications; its chief objectives include evaluating the scope and ramifications of security incidents, reproducing events, and providing relevant intelligence to aid decision-making or to aid legal proceedings.

In addition, an expansive definition of incident response begins with a clear definition of the word *incident*, which refers to a violation or a suspected violation of agreed-upon computer security policies or mutually agreed-upon security protocols. Thus, incident response is defined as the organized process used to handle and recover from a cyber assault. Examples that are pertinent to computer security incidents include botnets deluging a server with millions of requests to gain access, phishing operations carried out by email, unauthorized access to data by personnel, and extortion plots organized by hackers who demand ransom for public release of confidential organizational data.

Traditionally considered separate disciplines, digital forensics and incident response are increasingly complementary within real-world deployments. The complementary nature of these two fields highlights the importance of acquiring and scrutinising digital evidence efficiently and/or

near real-time as part of a holistic approach to security strategy. Computer Security Incident Response Team (CSIRT) is fundamental to managing and resolving all incidents affecting the security of the organization efficiently within a timely response period. The team is central to securing information and has the responsibility to maintain the tenets of Confidentiality, Integrity, and Availability (CIA) within organizational assets and sensitive data.

The digital forensics process is usually classified into four basic steps:

- **Identification**: The act of revealing pertinent digital evidence, such as files, data stored in RAM, network packets, application logs, etc.;
- **Preservation**: Capturing and saving data in a forensically sound manner, typically by using imaging or snapshot software;
- **Analysis**: Data examination using a variety of tools and approaches to discover relationships, identify anomalies, and reconstitute events;
- **Reporting**: A detailed document that communicates results clearly, succinctly, and within legal requirements.

There are standard models such as NIST SP 800-61 [6] or ISO/IEC 27035, that provide guidelines for developing incident response capabilities. Examples of those guidelines are preparation, detection, containment, eradication, recovery, and lessons learned, which together form the operational core of a modern CSIRT.

In order to ensure the legal admissibility of evidence, there is a need for collected materials to be carefully tracked within prescribed chains of custody. With modern forensic investigations, the application of timestamped acquisitions, the use of hashing functions like SHA256 to ensure data integrity, and establishing accountability with access logs are important requirements.

Incident response techniques increasingly emphasize the importance of forensic examination. Such examination provides an in-depth understanding of basic incident dynamics, identifies involvement of internal and external actors, as well as defines the scope of data that has been attacked. Gaining such insights at this juncture can significantly impact approaches taken for restoring systems, inform development of possible policies that can be used to forestall similar events, and aid compliance with legal and regulatory requirements.

In recent years, there has been a dramatic growth in the prevalence of sophisticated attack behaviors, like Advanced Persistent Threats (APTs), internal threats, and malware that exists only within memory or leaves very little residual evidence behind. They demand a shift in digital forensics from classic post-mortem examinations to an approach that focuses on stronger integration with real-time detection systems and automated collection tools for evidence. As such, there has been a great deal of focus devoted to the notion of **forensic readiness** [7]. Forensic readiness relates to an institution's ability to enhance capabilities within that institution.

One of the biggest challenges to achieving forensic readiness is managing volatile data. Volatile data that is present in system memory (RAM), network buffer, or temporary files can be easily lost by actions like rebooting or by log rollovers. Forensic professionals counter this problem by using live forensic tools like Volatility, along with collection techniques that are aimed at making data acquisition possible from systems that are operational. Systems need to be always pre-configured to support non-intrusive and safe data capture whenever possible.

Another important challenge relates to the mismatch between response times. Classical forensic examination can take a few days to a few weeks to deliver decision-ready results. In comparison, attackers can acquire sensitive data or clear logs within minutes. The comparison underlines why automation is so important by combining forensic techniques with monitoring ones. Organizations can start to gather evidence as soon as an attack occurs, thus collapsing the interval between the breach and subsequent response.

Digital forensics is central to ensuring compliance with regulatory standards and legal enforcement; regulations like GDPR, HIPAA, and PCI-DSS mandate that organizations present evidence for data breach incidents as well as subsequent response activities. Denial of access to forensic

data makes this process exceedingly difficult. The need is particularly acute within industries like defence and manufacturing. Downtime incidents, acts of sabotage, or unauthorized data access pose a potential threat to national security. Thus, there is a need to ensure that digital forensics works efficiently within standalone/air-gapped networks that are not connected to cloud resources or remote support systems. The need highlights an equally important requirement to develop forensic readiness to bolster response capacity, along with an offline response plan.

In such conditions, forensic preparedness should function independently, without aid, to serve for proper collection, preservation, and correlation of evidence at the scene; this often requires compliance with robust data export policies set by national security authorities. New developments around this matter have underlined a need for various specializations to intersect, especially those that are connected to forensic professionals, admin systems professionals, and info security professionals. The unifying of monitoring procedures with forensic techniques not only helps to create instant containment actions but also guarantees proper treatment of evidence, as illustrated by Eoghan Casey [8]. The unification acts as a basis for using all-inclusive tools aiming to integrate investigative as well as detection procedures; one such tool later to be analyzed within this paper, is Zabbix. As such, incident response and digital forensics function ever more reciprocally. The two specialisms are reliant upon one another to ensure that response to the issues presented by cybercrime occurs quickly, efficiently, and within legal bounds.

2.2 Network Analysis and Digital Forensics Tools

Network analysis is a cornerstone of digital forensics and is at the heart of understanding and examining cybersecurity attacks. As more advanced attackers utilize networks to assault systems, move laterally, exfiltrate data, or manage command-and-control (C2) operations, the ability to visualize, collect, and analyze network traffic is the pillar of adequate detection and subsequent incident investigation.

Network evidence can be examined on different levels. Raw packets provide the most information, flows offer useful summaries, and protocol analyzers extract important context. Each level provides unique insights. Using all these tools together helps investigators create a complete picture of a security incident.

Network forensics involves recording, monitoring, and analyzing network traffic to identify patterns of malicious activity or to reconstruct the sequence of events in a security incident. It is different from endpoint or disk-based analysis, which focuses on communication between systems - who sent what, when, and to whom. This is critical in cases of data theft, denial of service (DoS) attacks, or advanced persistent threats (APTs). Network traces provide crucial evidence of lateral movement or attempts to steal data.

Network forensic analysis usually begins by collecting packet data or logging flows at key network points, such as routers, firewalls, or taps. The main aim is to create a timeline of network activity that can reveal anomalies, unauthorized access, and potentially harmful communications.

The core of network forensics is the ability to capture and store network packets securely and in a way that maintains their integrity. Wireshark is the standard tool for capturing and analyzing packets. It offers a detailed graphical interface for looking at packet-level information, with features for protocol decoding, flow reassembly, and advanced filtering. Wireshark is great for small-scale analysis or for diving deep into specific flows.

In larger systems where capturing all packets is not feasible due to high data volume, command-line tools like tcpdump and dumpcap are used for capturing traffic at high rates. These tools allow users to apply custom filters and integrate scripts to capture traffic based on set criteria. The captured data is usually stored in PCAP format, which is the standard for archiving and later analysis.

For packet capture evidence gathered using packet capture equipment to be admissible in later forensic analysis, it needs to comply with strict technical requirements. First, accurate and synchronized clocks, generally via NTP, need to be used for information gathered to be successfully ordered and correlated across systems. Second, cryptographic hash functions like SHA-256 need to

be used upon capture for integrity preservation and indication of tamper detection. Secure devices with limited access, role-based authorization, and tamper-evident logs for auditing need to be used to provide accountability and traceability. Such basic controls are needed for preservation of the chain of custody and for ensuring network-based evidence will be able to stand up to challenge in legal, regulatory, or disciplinary proceedings.

While full packet capture yields maximum visibility, it is expensive and typically not feasible for long-term storage. More practical and scalable is flow-based monitoring, which aggregates packets into communication sessions or *flows*. Tools like NetFlow [9], IPFIX, and sFlow provide a summary of network interactions, including source/destination IPs, ports, protocol used, number of packets, and bytes transmitted.

Flow information can also be useful for identifying anomalies such as out-of-pattern data transfers, beaconing behavior (characteristic of malware C2 channels), or traffic to/from untrusted geo-locations. These tools are typically installed at core routers and serve as an early-warning indicator of compromise.

Open-source flow collectors such as nfdump, ntopng, and YAF (Yet Another Flowmeter [10]) can be combined with other monitoring solutions to provide alarms and facilitate historic probing. Flow records do not include payload information but are highly valuable in pattern detection and enabling richer analysis where needed.

While packet and flow data give an underlying perspective, protocol analyzers allow the extraction of semantic meaning from traffic that is captured. Zeek, formerly known as Bro, is among the most sophisticated tools within this class. It is a comprehensive analysis engine, reading network traffic in real time and logging for numerous protocols like HTTP, DNS, SSL/TLS, FTP, and SMTP [11].

Zeek doesn't inspect packets but instead generates detailed logs of session contents and metadata. As an example, for an HTTP session, Zeek can log the requested URL, the user agent, server response, and response time. These are easier to query and correlate than packet contents and are also most appropriate to forensic timelines.

Modular design of Zeek supports the integration of custom scripts and detectors, which further enhances its capacity to identify suspicious activity, policy contraventions, and application misuse. Zeek logs are usually inspected by log aggregation platforms such as the ELK Stack or Splunk to enable correlation with system logs and security alerts.

To allow qualitative analysis of data, dashboards and visualization tools like Kibana or Grafana are used by various organizations to track traffic patterns, examine anomalies, and inform decision-making during response. These platforms promote situational awareness and enable stakeholders who are less technically inclined to interpret forensic results.

The field of network forensics is greatly aided by the introduction of context to data that has been collected. Tools that map IP addresses to geolocation, WHOIS information, domain authority, or threat intelligence can take simple traffic captures and turn them into actionable information.

For example, Passive DNS databases enable the tracking of domain history across time intervals, Threat intelligence platforms label IPs and domains, and GeoIP scans identify traffic originating from high-risk locations.

Enrichment tools are particularly useful when traffic is heavy and prioritization is necessary. They are often used alongside Security Information and Event Management (SIEM [12]) systems for correlating network events with known attack indicators.

The value of network data relies not just on its collection, but also on its preservation. Interestingly, they are being integrated into automated detection pipelines, where anomaly-based events can lead to packet capture, log enrichment, or evidence archiving in near real time. This is important in time-sensitive investigations and aligns with the principles of forensic readiness.

Proven best practices dictate that:

- **Access to forensic data is strictly controlled and logged**, with role-based access control (RBAC), multi-factor authentication (MFA), and detailed audit trails;

- **Metadata is comprehensively recorded** for every capture, including cryptographic hash (e.g., SHA-256), precise source and destination context, capture location, and timestamps;
- **Retention policies are enforced according to legal, regulatory, and operational requirements**, ensuring that forensic artifacts are preserved for the appropriate duration;
- **Captured traffic is routinely validated for integrity**, with scheduled hash verification tasks to detect potential storage corruption or unauthorized modifications;
- **All forensic procedures are documented and reproducible**, enabling traceability and facilitating audits or third-party verification.

This verifies that network captures can be produced as admissible evidence in the courtroom or for disciplinary hearings, and that the forensic process is retained throughout the full duration of the investigation process.

The more difficult it is to discover cyber attacks and the more valuable it is to respond fast, network forensic tools need to be able to integrate with real-time detection systems. Flow analysis, protocol inspection, and traffic capture allow us to easily observe active attacks. Integrating forensic tools into live monitoring systems enhances the ability of organizations to respond and remain accountable. The process is proactive and ensures valuable evidence is gathered, recognized, and reserved for further analysis.

2.3 Network Monitoring Solutions in Cybersecurity

Network monitoring also plays a central role in the field of cybersecurity because network monitoring enables staff in an organization to monitor the activities of devices, services, and users in a networked setting [13]. As opposed to the use of forensic tools following the occurrence of an incident, monitoring systems are in constant use and provide real-time feedback about the state of the IT infrastructure.

The network monitoring tools can also be classified broadly into three categories according to their application and analysis depth:

- Infrastructure Monitoring Tools - These tools emphasize operational measures like CPU usage, memory capacity, disk storage, and service availability. These tools mainly find application in system availability management and IT performance.
- Intrusion Detection/Prevention Systems (IDS/IPS) - They check network traffic for predefined attack patterns or behavioral anomalies and are capable of triggering actions taken for warning or blocking.
- Security Information and Event Management (SIEM) Systems - The SIEMs collect and parse logs from many different sources and apply rules to detect complex patterns of attacks and generate alerts.

Each of the devices plays a different function in the detection and response, thus it can be integrated into a multi-level security infrastructure.

System infrastructure monitor software such as Zabbix, Nagios, and Prometheus aggregate real-time metrics from servers, network gear, applications, and services. The platforms themselves are not security-specific but can be reassigned for monitoring for unusual behaviors indicative of a compromise, such as:

- Isolated spikes in network or CPU activity;
- System or service restarts on a routine basis;
- Sudden login attempts or failed authentication;

- Elevated disk activity or abrupt log file increase.

The above indicators, once established as alerts and merged within automation scripts, support the timely identification of potentially malicious activity. Zabbix, for example, allows one to create triggers that activate upon thresholds being exceeded: such responses may include notification emails, running of scripts, or log events. In this way, infrastructure monitoring serves as a first layer of visibility and proactive handling in the security framework.

For Operational Technology settings, such as manufacturing or infrastructure, the application of endpoint agent or common antivirus software is often constrained by outdated systems or the necessity for real-time capabilities. In such a case, the application of infrastructure monitoring software becomes relevant for the identification of early anomalous activity, such as unusual CPU or network activity occurring between controllers. Such anomalous activities can act as early indicators of compromise in systems normally hard to monitor or secure by common security applications.

Intrusion Detection Systems (IDS), including Snort [14], Suricata [15], and Zeek, are employed to scrutinize network traffic for the identification of known threats. These IDS function passively by generating alerts, whereas Intrusion Prevention Systems (IPS) possess the additional capability to actively obstruct traffic. Signature-based IDS demonstrates considerable effectiveness in detecting specific threats, such as malware payloads and port scans; however, it frequently encounters challenges with zero-day or stealth attacks. In response to this limitation, a variety of platforms have begun to integrate anomaly detection, heuristics, and behavioral models. For example, Suricata facilitates multi-threaded analysis and protocol-aware inspection, rendering it appropriate for high-performance operational contexts.

A combination of the alerts from the IDS/IPS and the SIEM or log management systems provides improved forensic traceability and cross-correlation in the user or system events. Security Information and Event Management (SIEM) products provide a centralized view of the security posture of an organization. According to NIST SP 800-92, a SIEM should collect, normalize, and examine security-relevant information from various sources in support of incident identification and handling [16]. Products like Splunk, IBM QRadar [17], Elastic Stack (ELK) [18], and LogRhythm collect logs from firewalls, servers, authentication servers, endpoint detection tools, and so forth.

Contemporary Security Information and Event Management (SIEM) platforms typically provide real-time notifications triggered by breaches of predefined rule sets or the surpassing of established risk thresholds. Furthermore, these platforms facilitate integration with external threat intelligence sources to enhance detection capabilities and employ advanced analytical techniques or machine learning algorithms to identify anomalies that might otherwise remain unnoticed. Additionally, these systems support extensive data indexing necessary for retrospective forensic investigations and adherence to regulatory compliance requirements. Despite their considerable flexibility, SIEM platforms frequently exhibit complexity and demand significant resources for both initial deployment and ongoing operation. Skilled personnel are required to establish correlation rules, refine log ingestion procedures, and ensure that appropriate data retention protocols are effectively implemented. To improve certain operational aspects, some organizations have begun to utilize Security Orchestration, Automation, and Response (SOAR) systems that automate and standardize response procedures [19].

Each monitoring solution type serves a distinct role in the detection pipeline: continuous monitoring allows for real-time visibility of system and operational performance, IDS/IPS tends towards inspection of network-level behavior, SIEM allows for the correlation and long-term analytics across the entire infrastructure.

In practice, the interaction of these systems forms a robust defense-in-depth approach. Zabbix, for example, sees a suspiciously high CPU usage, Suricata sees malicious packets concurrently, and a SIEM sees the two and recognizes a series of unsuccessful logins, confirming a brute-force attack in progress.

Despite not being specifically created as forensic instruments, monitoring systems are vital in facilitating forensic preparedness. Their contribution is evident in the following ways:

- Assuring the storage and logging of telemetry and logs;
- Initiating alerts for deeper forensic collection;
- Preserving context and metadata, which are significant in constructing the timeline;
- Supporting legal and regulatory investigations by maintaining event traceability.

Network monitoring products act as the monitoring elements of a security system in an organization and reveal the first signs of compromise while also allowing for swift, evidence-led responses.

Furthermore, as this work examines, the use of tools such as Zabbix not only gives more insight into operations but also forms the basis for forensic readiness. With the right configuration, Zabbix can also be incorporated into automation for the collection of evidence and therefore cause the advent of a security event to trigger forensics recording and alerts in near real time [20].

2.4 Comparative Analysis of Monitoring Solutions

The topic of forensic preparedness has attained ever-broadening prominence among both technical research studies as well as industrial cybersecurity practice. Organizations that run under fast-paced, highly complex, and unpredictable cyberattacks require methods for reducing mean time to detect (MTTD) as well as mean time to respond (MTTR), while keeping evidence handling forensically sound. There exists an increasing collection of work combining monitoring, detection, as well as automated evidence collection intended for filling the gap between operations (SOC) as well as investigations (DFIR). This section broadens the comparison among Zabbix as well as competing methods, incorporating open-source NIDS/HIDS stacks, enterprise SIEM/SOAR platforms, as well as technical prototypes for automated, trigger-based forensic collection.

Academic Prototypes and DFIR Frameworks

Many of the research studies propose systems reacting to indicators of compromise (IoCs) or abnormal behavior by *immediately* collecting volatile and non-volatile evidence.

AutoForensics Li et al. [21] outline AutoForensics, an architecture for reacting to suspicious activity by capturing system state, memory dumps, and metadata snaps at near-real-time latency. Tamper resistance through the use of hashing and chain-of-custody logging takes top priority for the system, and supports SIEM-driven triggers as well in an effort to minimize lag between detection and collection.

Cloud Forensic Readiness Marturana et al. [22] suggest a modular forensic readiness framework for the cloud that initiates capture workflows in response to IDS alert events, filling evidence volatilities characteristic of cloud-native infrastructure (e.g., transient instances, containerized workloads).

DFIR for OT/Critical Systems The NIST DFIR methodology for Operational Technology (OT) [23] codifies industrial incident handling where the safety and availability requirements dictate policy-guided, minimally intrusive capture. It completes the industrial monitoring stacks by prescribing when and where forensic capture must take place without disrupting mission-critical operations.

Automation in Digital Forensics The domain has reached a consensus on more precise definitions and boundaries regarding automation [24], facilitating tiers that encompass scripted tasks to entirely automated, event-driven pipelines. This body of literature emphasizes the necessity of integrity, reproducibility, and legal admissibility as essential alongside efficiency and comprehensiveness.

Industry-Grade Stacks (SIEM/SOAR, HIDS/NIDS) and Open Ecosystems

NIDS/HIDS exemplars (Zeek, OSSEC/Wazuh) Zeek excels at rich protocol-level telemetry and retrospective reconstruction but doesn't natively provide cryptographic sealing or chain-of-custody beyond log controls [11]. OSSEC and its continuation, Wazuh, offer host-based intrusion detection, file integrity monitoring, and central management, commonly utilized as an aid for forensic triage and post-incident examination. Wazuh implementations showcase efficient centralized log management as well as selective investigations in practice [25, 26, 27].

Open-source SIEM platforms (Elastic/ELK) Elastic Stack enables scalable log ingestion, correlation, and anomaly detection led by ML, as well as acts as the analytics engine for open-source SIEM product offerings [26]. Forensic integrity components are inherent to it (cryptographic sealing of captured images, for example). However, tend to require add-in tooling or custom pipelines.

Enterprise SIEM/SOAR (Splunk, QRadar, Elastic Security) Enterprise SIEM/SOAR offers playbooks and automation of responses to orchestrate acquisition steps on triggers (behavioral and rule-based). Surveys point to their depth but also point to operational complexity and resource requirements as being too high in tight OT or air-gapped applications [28, 26].

Blockchain-secured Evidence Integrity and Chain of Custody

Recent works explore the use of blockchain for enhancing chain-of-custody auditability. Solutions range between smart-contract-based access controls and immutable evidence metadata to IoT investigation-specific architectures [29, 30]. They aspire for transparency, non-repudiation, as well as cross-organizational trust, but ISO/IEC 27037 integration and scalability remain areas of future work.

Positioning Zabbix for Automated Forensic Readiness

Although its primary development was not for forensic purposes, Zabbix presents a trigger-action system for the purpose of orchestrating **external** forensic acquisition processes when exactly defined anomaly states occur. In the architecture proposed in this dissertation, Zabbix triggers invoke scripts first, saving volatile memory or critical runtime states as snapshots, then logging and timestamping data, and finally starting restricted network capture sessions.

By embedding hashing, timestamping, and chain-of-custody logging within those scripts, forensic integrity may be added to Zabbix workflows. It maintains the strengths of Zabbix-lightweight deployment, agent/agent-less flexibility, heterogeneity support-whilst delivering DFIR-compliant acquisition at the time of detection. Compared to enterprise-caliber SIEM/SOAR, Zabbix requires more custom script work for forensic-caliber documentation and integrity assurance; compared to research prototypes with built-in integrity modules, Zabbix prioritizes extensibility at the sacrifice of vertical completeness. It's the usual trade-off of choice for resource-limited or legacy OT environments, air-gap networks, and heterogeneous mixes where operator control and footprint matter.

Comparative Tables

Table 2.1. Academic Proposals and Forensic-Readiness Features

Proposal/Framework	Trigger-based	Integrity & CoC Mechanisms	Primary Contexts
AutoForensics	Yes	SHA-256 hashing, automated chain-of-custody logging, tamper-proof storage.	Enterprise, real-time incident response.
Cloud Forensic Readiness	Yes	Workflow-driven integrity controls, cloud-native evidence handling.	Ephemeral workloads, container forensics.
NIST DFIR for OT	Policy	Process-level integrity guidance, safety-constrained acquisition.	Critical infrastructure, industrial control.
Automation in DF	Varies	Reproducibility emphasis, validation frameworks.	General DFIR methodology.
Blockchain CoC	Event	Immutable metadata, smart contracts, distributed trust.	IoT investigations, multi-org cases.

Table 2.2. SIEM/Monitoring Tools and Forensic Capabilities

Tool/Stack	Trigger-based	Forensic Integrity Features	Deployment Context
Zabbix	Yes	External scripting for hashing, timestamps, custody logs.	IT/OT, air-gapped, legacy systems.
Zeek (Bro)	Partial	Log-level controls only, no cryptographic sealing.	Network-centric monitoring.
OSSEC/Wazuh	Yes	File integrity monitoring, limited native CoC.	Enterprise/SME, some ICS deployments.
Elastic/ELK SIEM	Yes	Pipeline extensions for integrity, no native imaging seal.	Large-scale IT, hybrid cloud.
Enterprise SIEM/SOAR	Yes	Audit trails, case management, variable imaging integrity.	Enterprise, cloud, MDR services.

Trade-offs and Complementarity

Academic proposals such as AutoForensics prioritize end-to-end forensic soundness with low-latency capture and built-in integrity and custody controls [21]. Frameworks for OT environments emphasize policy-constrained acquisition that respects safety and operational continuity [23]. Enterprise SIEM/SOAR delivers orchestration breadth but at higher complexity and resource cost [28]. Open stacks (Zeek, OSSEC/Wazuh, Elastic) are modular and extensible, yet typically require additional engineering to meet strict forensic admissibility thresholds, especially for cryptographic sealing and formal chain-of-custody [26, 11, 27].

Zabbix sits at a pragmatic midpoint: a lightweight, scriptable monitoring platform that can trigger external acquisition and integrity workflows at the precise time of detection, which is critical for volatile evidence. In constrained or heterogeneous environments, this flexibility and low footprint can outweigh the absence of built-in forensic modules-provided that acquisition scripts enforce hashing, timestamping, custody logs, and storage hardening. In this sense, Zabbix can act as the forensic readiness orchestrator in a pipeline that satisfies DFIR expectations.

Summary

Although Zabbix itself does not provide native forensic sealing or formal chain-of-custody, it makes up for it with extensibility and accurate trigger-action orchestration on disparate and resource-limited hardware. Through the combination of scripted hashing, timestamping, custody logs, and secure storage, Zabbix affords an affordable and technically possible basis for automated, incident-based evidence collection. It complements research papers focusing on legal soundness and enterprise SIEM/SOAR, providing orchestration breadth, making Zabbix a resilient focal point for forensic readiness in mixed IT/OT environments.

Chapter 3

Automated Forensic Data Collection Model with Zabbix

This chapter outlines the primary contribution of this thesis: an exhaustive architecture intended to employ Zabbix as a vehicle for the acquisition of real-time digital forensic evidence. Complementary to the comparative evaluation provided by Chapter 2, which emphasized the potential advantage of Zabbix compared to commercial SOAR offerings with respect to flexibility and economy, this chapter moves from theoretical discussion to practical architectural design.

The organizational structure seeks to guide the reader through a systematic process, from theoretical foundations to implementation readiness. The first part discusses the fundamental nature of Zabbix, applying a forensic approach to demonstrate how standard infrastructure monitoring functionalities can be reworked to serve evidential needs. This discussion is guided by the constraints identified in Section 2.4, i.e., the lack of inherent forensic integrity features in Zabbix, but it makes up for these by taking advantage of its established strengths in trigger-based and programmatic behavior.

Next, a modular architecture is introduced that divides the process of collecting forensic evidence into four discrete but related components: detection, response, acquisition, and preservation. The architecture complies with the separation of concerns principle, aiming to increase the resilience of forensic systems; in addition, it allows for the inclusion of external tools to overcome the limitations faced when using Zabbix in isolation. Every module is designed to be self-contained; however, they are linked in the shared working system of Zabbix.

The final parts of this chapter analyze the main challenges that come with trigger design, minimizing false positives, and the automation of workflow orchestration. Special attention is given to the technical dimensions and design features that need to be taken into account when adapting a multifunctional monitoring device for forensic use. The analysis wraps up by assessing the scalability and flexibility of the framework, thus providing grounds for the empirical studies that will be discussed in the next chapters. The thorough analysis justifies the theoretical significance and practical applicability of the proposed approach, showing potential for leveraging the architectural flexibility of Zabbix in a bid to create an affordable alternative for sophisticated automated forensic systems.

3.1 Conceptualizing Zabbix for Forensic Readiness

In order to explore the possibility of Zabbix in providing automated forensic readiness, it is critical to firstly grasp the inherent mechanisms supporting its monitoring capabilities. This section examines the traditional monitoring capabilities, such as metric aggregation, anomaly detection, and information retention, and how these would be improved to satisfy evidentiary standards. The focus is placed on the approaches through which data integrity, timestamp validation, and automated reaction can provide a premise for a verifiable forensic collection process.

3.1.1 Continuous Monitoring to Tamper-Proof Storage

Zabbix has been used conventionally in mission-critical environments for real-time monitoring of IT infrastructure health, availability, and performance. Its modularity and adaptability have made it an effective tool for identifying anomalies and notifying system administrators in real time. In its essence, Zabbix works by collecting information from monitored hosts via agents or SNMP and matching these metrics with user-defined thresholds. Upon deviation of a metric from expected operations, Zabbix generates a trigger and performs set actions. This provides for quick identification of faults and assists organizations in meeting service-level agreements and maintaining the stability of infrastructure.

Nonetheless, the same principles that allow for high-availability monitoring can be used for forensic purposes. Since Zabbix captures and stores metric history in a structured and timestamped format, it naturally creates a partial audit trail of system activity. Combined with item-level historical logging, actions, and external scripts, Zabbix can be set up to act as a real-time sensor to trigger forensic evidence collection. For example, an unexpected network traffic spike or failed login attempts can be indicators of compromise, and Zabbix can be set up to instantly run a logging script or start packet capture.

Zabbix's local database, in addition, (usually based on MySQL or PostgreSQL) provides robust, tamper-resistant logging of system events. While not a tamper-proof archive in itself, its support for external log signing and hashing mechanisms allows the construction of a pipeline that extends from real-time monitoring to secure and verifiable storage. This proves particularly significant in contexts where events must be preserved for legal or regulatory reasons, in cases where near-immediate visibility into system activity can improve both incident response and subsequent forensic analysis.

3.1.2 Why Zabbix is Suitable for Real-Time Evidence

The migration from being a largely reactive monitoring tool to becoming a core element of a forensic incident response framework relies mainly on its modular and extensible design. An important factor that enables this extensibility is its event-driven nature, as seen in Zabbix. This aspect not only allows for accurate data collection but also allows for the running of scripts based on user-provided parameters. Unlike traditional post-event forensic methods, which are largely based on the retrospective analysis of logs, this integration enables evidence collection at the time of its detection, while maintaining its volatility and contextual integrity.

Before delving into the analysis of Zabbix's forensic capabilities, it is essential to understand what the main components of the architecture are and how they interact with each other. The Zabbix distributed monitoring system is divided into four main elements: the Zabbix server represents the central processing node, which is responsible for evaluating the activation conditions and storing historical data within a backend database, generally based on MySQL or PostgreSQL.

Zabbix agents, installed on monitored hosts, are processes that deal with the collection of local statistics and the execution of authorized remote commands. Zabbix proxies, on the other hand, operate as intermediate collectors in distributed configurations, with the goals of reducing network load while ensuring the monitoring of isolated network segments.

Finally, the web frontend provides the interface for the overall configuration, visualization, and management of the monitoring system. Communication between these components can occur in active polling mode, where the server queries agents, or via passive trapping, where agents proactively send data to the server; both exchange mechanisms are protected by PSK (Pre-Shared Key) or certificate-based encryption systems.

Zabbix adopts a decentralized architecture, composed of agents, proxies, and a central server, whose coordination takes place via polling or trapping-based communication mechanisms. Data collected by monitored hosts is stored in a backend database and made available through a front-end web interface. Zabbix agents, installed on endpoints, collect and transmit system performance information at user-configurable intervals. This data, called elements, is processed by the server evaluation engine, which simultaneously checks for the occurrence of predefined conditions. When

a trigger returns a positive value, Zabbix can react, such as sending notifications or running automated scripts.

The inherent flexibility that is built into Zabbix's architecture makes it usable in forensic environments. Detection of abnormal activity, like unauthorized network access, unexpected spikes in CPU loads, or file integrity breaches, can be used as triggers. The triggers then run external forensic scripts to gather volatile data, create file hashes, or take system snapshots. Zabbix's ability to be used with nearly any scriptable process makes it a solid bridge between real-time monitoring and forensic analysis.

Figure 3.1 outlines the general architecture of Zabbix together with its principal components, namely the Server, Database, Web Frontend, Proxies, and Agents. Elaborating on this infrastructure, Figure 3.2 illustrates a schematic view of the logical relationships between these components within an architecture to support forensic functionality, showing how standard monitoring techniques can be extended to allow for the automatic gathering of evidence.

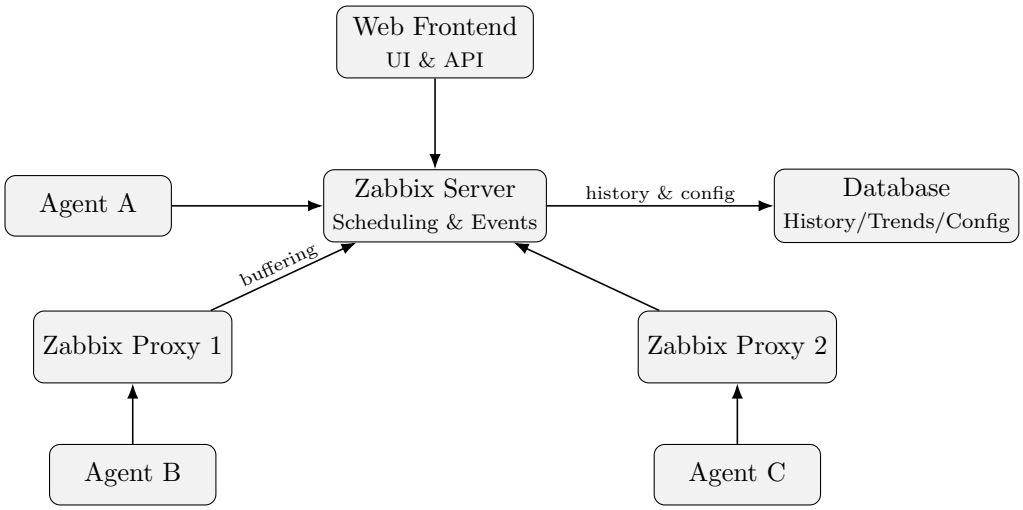

Figure 3.1. General Zabbix architecture: Server, Database, Web Frontend, Proxies, and Agents

This architectural flexibility directly addresses the deployment constraints identified in Section 2.4, particularly Zabbix's suitability for "IT, OT, Air-gapped, Legacy" environments as documented in Table 2.2. Especially beneficial are situations where conventional Security Information and Event Management (SIEM) products or Endpoint Detection and Response (EDR) technology are not practical because of operational constraints or security considerations. Industrial Control Systems (ICS), stand-alone tactical operations, and aged infrastructures often lack the resources or privileges needed to apply security products requiring considerable system resources. In these environments, Zabbix presents an advanced but flexible tool that, when implemented with minimal disruption, can be customized with security-oriented functionality.

Furthermore, the structured format of item polling in Zabbix, combined with its extremely flexible trigger expressions, ensures accurate detection specifically crafted according to the environment under supervision. The `count()`, `avg()`, and `nodata()` functions permit temporal and statistical correlation between disparate metrics, while macros enable a dynamic method of conveying contextual information into scripts. All this, coupled with secure storing of evidence along with hash verification procedures, makes Zabbix an integral tool for preserving data integrity and securing the chain-of-custody throughout the lifecycle of the incident handling process.

The core assumption is that forensic preparedness need not be viewed as the creation of entirely new solutions, but as a rigorous examination of how existing monitoring tools, like Zabbix, can be used to greatest advantage. Properly configured and armed, Zabbix becomes more than a simple telemetry collector and instead becomes a watchful forensic guard, making it possible to collect evidence in near real-time, the gap between detection and legal defensibility thus being closed.

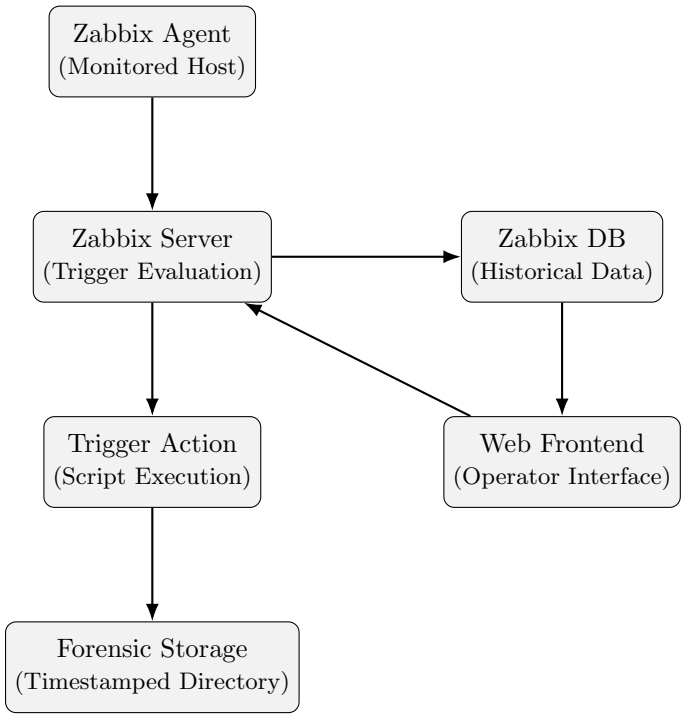

Figure 3.2. Logical flow of forensic-enabled Zabbix architecture

3.2 Modular Architecture for Evidence Collection

3.2.1 Overview of the Proposed Architecture

The outlined approach shares more with modular architecture than with a monolithic system, in that individual modules have clearly defined functions in the process of evidence collection. Since the modular approach increases this process's reusability, scalability, and maintainability can be upgraded, and examined with little danger to the stability of the pipeline as a whole due to defined responsibilities. Communication between the modules takes place through Zabbix's action system and therefore guarantees cohesion and consistency between these units in the process.

The underlying assumption behind the conceptual design proposed in this research states that the forensic evidence collection needs to be performed neither in a fragmented and reactive spirit nor in isolation, but should be integrated within an automated and organized procedural system. Zabbix presents a useful template for aggregating this pipeline of evidence through its user-defined action and trigger functions.

The **Detection Module** recognizes suspicious or possibly dangerous activity within the observed infrastructure. Although using common Zabbix triggers and items, the focus of this module's logic lies in the detection of patterns that are characteristic of security events, more than ones that are related to performance-based situations. Detection thresholds can be either temporal (e.g., repeated failed efforts to log in within 30 seconds) or metric-based (e.g., anomalous usage of listening ports). These triggers are the primary means to stimulate the process of gathering forensic information.

When an anomaly is discovered, the **Response Module** is triggered. This module encapsulates the inherent logic of the Zabbix Action, defining the relevant commands to issue in response to defined triggers. It allows for custom scripts to be run locally (on the host) or remotely (through SSH or agent commands). Here, the response process doesn't actively alert or resolve incidents; it serves more like a dispatcher that triggers scripts with the scope of evidence gathering. Execution process ensures preservation of contextual information and reduces redundancy in execution, done through escalation levels or cool-down timers.

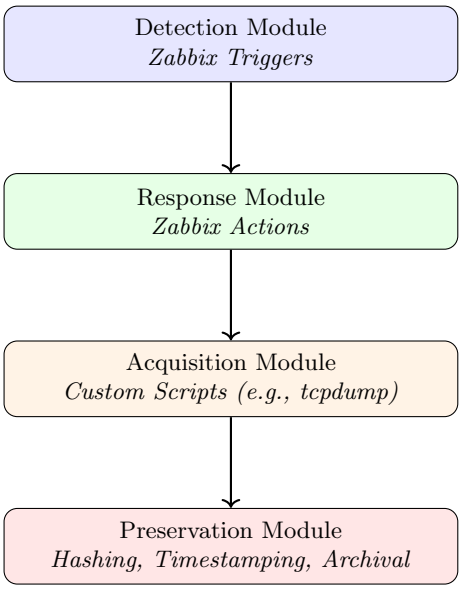

Figure 3.3. Modular Workflow for Automated Forensic Collection using Zabbix

The **Acquisition Module** comprises all scripts and procedures required for acquiring evidence that qualifies as either semi-volatile or volatile. Based on the incident categorization, this module can perform different actions like invoking a memory dump through the usage of external tools, invoking a packet capture (e.g., using *tcpdump*), capturing live process trees, gathering certain extracts from the log base, or listing open network connections. Each of these scripts executes atomically and is constructed to prevent data contamination. Generated outputs are always written to a temporary secure directory that utilizes a special name convention derived from a timestamp and hash identifiers, making traceability easier to accomplish.

The last operation is performed by the **Preservation Module**, calculating a hashing function for all gathered data (e.g., using SHA-256) and embedding timestamp information. This module ensures that evidence remains securely saved, encrypting sensitive elements if required, and stores it in a format that will only allow additive updates. In addition, it keeps a thorough history of the chain of custody record, recording information about the individual who gathered data, the gathering time, the situations that required data gathering, and any scripts that were run. With proper configuration, the module can handle moving evidence artifacts to remote forensic vaults or secure remote storage areas. Note the intentional difference in logic between preservation and acquisition: the former emphasizes rapid data extraction, whereas the latter emphasizes its appropriate legal and procedural protection. Post-event forensics and legal admissibility are eased during required usage through the architecture.

3.2.2 Interaction Between Zabbix Modules

Incorporation of the four advanced modules (Detection, Response, Acquisition, and Preservation) follows an ordered but flexible format that takes cues from Zabbix's internal macro expansion, runtime logic, as well as escalation plans. Once an anomaly has been detected, the trigger records the event but also fires an action that runs an orchestrated command. These scripts act per parameter drawn from the environment, sent via Zabbix macros, thus ensuring each action remains properly contextualized and documented with the date and time.

The structure supports synchronous and asynchronous execution of operations. For example, acquisition scripts can finish before preservation routines are initiated, or else they can execute simultaneously in case a decrease in latency is necessary. The mechanism of synchronization is managed by Zabbix's native dependency management and escalation policies. These policies allow the creation of conditions under which the execution of actions is triggered, define retry periods, and introduce fallback mechanisms.

Under this architectural model, the action mechanism of Zabbix becomes the unifying factor that connects the different modules, thus enabling a consistent response pipeline. At the end of each of these stages, execution is transferred to the next one through systematically organized books and by recording the output. This way, the evidence lifecycle proceeds seamlessly: detection triggers the collection process, capture scripts capture volatile or semi-volatile data, and retention features enforce integrity mechanisms, ensuring secure artifact storage.

Unlike other, more centralized digital forensic suites like traditional SIEM/SOAR integration, the modular platform takes advantage of the strengths of "adaptability, simplicity, and automation," as described in Section 2.4, thus offering increased performance with lower system overheads along with increased portability. For example, solutions like Splunk, when SOAR-integrated [28], support end-to-end forensic orchestration; however, these tend to require proprietary connectors along with complex configuration processes that are difficult to use. In contrast, the Zabbix-based approach follows the basic tenets defined in forensic guidelines like NIST SP 800-101 [31], which include automation, integrity checking, and non-interference of running system processes. In addition, this modular approach reduces the effect of individual failures; individual modules could be isolated, inspected, and improved independently. The use of customized scripts, with version control and auditing procedures as complements, dramatically increases traceability, an important factor in ensuring legal compliance. Accordingly, in small deployments with limited enterprise-wide solutions, Zabbix presents an adaptive, non-obtrusive substitute that ensures adequate forensic readiness.

By synchronizing, in real time, the forensic pipeline with stimuli and enabling script-driven execution with an intention toward evidence collection and preservation, this system achieves its forensic goals while, at the same time, respecting fundamental principles of system resilience and maintainability. The interactivity among the modules goes beyond typical functionality, with an obvious design choice that's best aligned with current best practices in building forensic systems.

3.3 Triggering Mechanisms and Custom Logic

3.3.1 Designing Triggers for Forensics

The proposed modular design includes the *Detection Module*, which leverages the trigger facility inherently found in Zabbix to initiate the forensic evidence pipeline. The trigger found in Zabbix serves as a logical function that evaluates monitored item values, changing its status to problem when certain conditions are met. In order to ensure forensic readiness, it is important that these triggers be configured not only to monitor operational health but also to recognize special patterns that indicate security breaches, thus allowing for timely and effective measures directed towards evidence gathering.

Unlike performance-focused monitoring, which largely evaluates service responsiveness and resource utilization, the purpose of the underlying forensic triggers is to correlate system-level measures with security-relevant indicators. Such correlation often means the aggregation of multiple telemetry sources - e.g., deviations from normal CPU performance, unusual invocations of processes, or sudden spikes in network traffic by destination - via the application of logical operators along with time functions. For instance, the Zabbix function `avg(/host/key,(sec|#num)<:time shift>)` can be set up to detect extended abnormal behavior within a given timespan, while the `count()` function can be utilized for identifying repeated patterns, for instance, multiple unsuccessful authentication attempts or multiple invocations of suspicious process IDs.

A key forensic observation tool is the use of the `nodata()` function, which indicates `true` when there is no data from a monitored entity within a given time interval. This feature helps identify possible log suppression or alteration by agents. For example, correlating the output of `nodata()` from audit logs with data related to system uptime helps trigger the generation of alerts when expected activity reports are absent, indicating that a logging service has been disabled or compromised. Triggers in Zabbix can have nested expressions, allowing complex logical conditions to better map onto real-world attack behaviors. As such, a forensic trigger can be configured to only trigger upon the combination of an escalation of network traffic out and the running of a

non-whitelisted process, greatly reducing false alarms when compared to single-condition-based rules. Such complex logic is especially relevant to detecting complex events like data extraction or lateral movement, where no single metric on its own is definitive.

Forensically guided trigger construction requires a delicate balance between sensitivity and specificity. Rules with high sensitivity produce a high number of false positives, while those with high specificity leave subtle indicators of compromise undetected for too long. The balance is best struck by iterative refinement, informed by historic baselines, tested with empirical evaluation under controlled circumstances. In this way, a structured approach could include the deployment of `prev()` and `change()` functions to pinpoint deviations from the baselines for normal operations, with rules to detect abnormal service restarts or anomalies with the user authentication behavior. The design of the triggers becomes, therefore, a dynamic aspect of the organization's forensic readiness architecture, responding to changes both within the infrastructure as well as the threat environment.

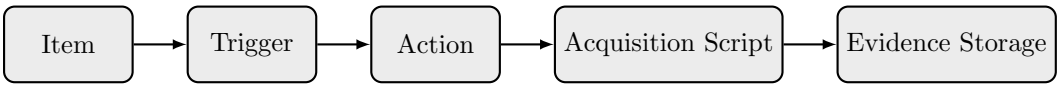

Figure 3.4. Compact logical flow from monitored item to preserved forensic evidence.

3.3.2 Reducing False Positives

Under the modular architecture that this thesis presents, the trigger logic incorporated inside the *Detection Module* has the essential task of determining the correct instances needed to trigger the acquisition and preservation phases. However, the major challenge within this process is the minimization of false positives. In the scope of forensic automation, the consequences of a misleading trigger are particularly significant, with every activation event starting up resource-hungry processes like memory dumps, network captures, or comprehensive log extractions. These processes not only require extensive computational resources but also come with the risk of overwhelming the analyst with unnecessary alerts, eventually leading to the loss of confidence within the system.

Zabbix offers several ways to do this with its handling of complex, contextual activation criteria. The availability of logical operators like `&&` and `||` allows for the combination of various independent conditions into a composite, high-confidence trigger expression that accommodates a wide range of diverse, independent conditions to include: the duration for which the CPU threshold would need to be exceeded, the absence, on a pre-determined whitelist of known legitimate applications, of the errant process, and the presence, on the host computer's network, of simultaneous suspicious network traffic. The resulting composite condition logic largely eliminates false events, like temporary spikes in workload or maintenance activity, thus focusing forensic analysis on events where multiple corroborating indicators imply potentially malicious activity.

Temporal functions enhance the effectiveness of noise reduction by separating transient irregularities from persistent deviations within normal behavioral patterns. The functions `delta()`, `trendavg()`, and `max()` can be used over variable time intervals to prevent transient irregularities, like spikes on system update during log-on, from being falsely identified as widespread forensic activity. This capability can be tailored to system roles through the use of user or host group macros, thus raising sensitivity for mission-critical servers while permitting more relaxed thresholds on endpoint systems that are low risk.

A further relevant method is to link incidents between different hosts. Zabbix implements the trigger dependency mechanism, which allows secondary triggers to be automatically deactivated when a related primary trigger is activated. For example, if an anomaly is detected at the perimeter of the entire network, triggers related to internal systems can be temporarily suppressed until the main problem is resolved, thus avoiding the production of redundant evidence. This hierarchical model of dependencies promotes priority setting and ensures that investigations focus on the aspects with the greatest impact.

Intrinsic escalation policies for action establishment can be a means to counter unwarranted disruptions. An incremental approach might begin with fairly non-intrusive investigative tasks,

like detailed log analysis or procedural listing, at the time a suspicious condition is detected, escalating to more intrusive methods, like packet capture or memory imaging, only if the suspect condition persists across successive testing cycles. A systematic escalation maximizes the effectiveness of resource utilization without compromising preparedness for pervasive evidence gathering when needed.

Context filtering, facilitated by the enforcement of whitelists and anomaly baselines, adds another level of sophistication. By modeling benign behaviors explicitly observed, such as scheduled security scanning or software deployment processes, the system can avoid misclassifying such events as incidents. Context sensitivity can be achieved by setting scheduled trigger suppression periods or by adjusting the threshold dynamically with the help of operational calendars. Finally, the reduction of false positives across the scope of forensic monitoring cannot be viewed as a monolithic configuration challenge but instead calls for an iterative optimization strategy. Desiderata obtained from actual activations should inform the constant tuning of those triggered on logic, thresholds, and correlation measures. In the scenarios where high confidence is required - e.g., operational technology networks for mission-critical infrastructure or radio silo-based defense systems - the incremental refinements enable the running of the forensic triggers with increased accuracy and reliability. Although the present analysis focuses on rule-based optimization, future studies could explore the integration of statistical anomaly detection or machine learning techniques with the Zabbix trigger analysis paradigm, as shown by other investigations on hybrid monitoring systems [32, 33].

3.4 Automated Response Workflow in Zabbix

3.4.1 Executing Scripts from Triggers

Zabbix supports a complex architecture that allows the implementation of automated responses through its *Action* feature, which allows the sending of remote commands or messages upon certain events. While this feature so far has mostly been employed in alerting system administrators via SMS or mail, it can be customized towards forensic applications through linking a trigger to a custom script that is written to undertake a succession of evidence-gathering measures.

With the implementation of this approach, one specifically effective method consists of setting up a custom *Media Type* or using the *Remote Command* feature. With either approach in mind, Zabbix will be prompted to run a given script either hosted on the remote host that will be targeted for monitoring or on a preselected remote collector node. This given script will be linked to a trigger via an action that was previously configured via the Zabbix frontend and/or API.

One significant benefit of this setup is the ability to integrate dynamic runtime parameters in the script. Zabbix macros like `{HOST.HOST}`, `{IPADDRESS}`, `{ITEM.VALUE}`, and `{EVENT.DATE}` can be woven in the command string, which will be replaced with relevant values related to the alerting event upon execution. This allows for some level of contextual understanding that's unique to the host and to the event, a functionality that's critical in automating forensic analysis. As a case in point, you might supply the IP address of a troublesome host to a script that you want to kickstart a packet capture session specifically for that endpoint. Upon running the script, it must be ensured that it functions properly and that it logs appropriately. Scripts should also return appropriate exit codes following POSIX guidelines, where `0` indicates successful termination, and any non-zero indicates error. Additionally, every run must produce log output, either locally or sent to a centralized log aggregator, with information about the timestamp, received parameters, and performed actions, including any errors that took place. Such output of the log plays a crucial role not only for debugging but also for maintaining the integrity of the forensic chain of custody and procedural reproducibility. Ideally, every action script will compute a hash function (e.g., SHA-256) for any produced artifact and retain the artifact and its respective metadata in an immutable manner.

To ensure that commands are run only on systems that hold explicit permission, you must set the `EnableRemoteCommands` parameter to `1` in the settings of the agent. We also advise you to adopt command whitelisting as a supplemental measure to prevent the commencement of unwanted and malicious activity due to improper settings.

3.4.2 Multi-Step Forensic Collection

Zabbix's ability to aggregate different response actions makes it suitable to manage a streamlined forensic acquisition process. Despite the lack of native support for complex workflows and the management of states, its scripting capabilities and event-based architecture make it act like a coordinator to collect evidence throughout numerous stages.

In a traditional setup, a system that identifies abnormal behaviors - in the form of unauthorized port scans, unusually bulk data transfers, or suspicious CPU usage - can trigger a series of actions through the medium of a shell script. This shell script, in turn, triggers a series of commands specifically designed to systemically and logically collect forensic evidence in real-time. For example, the script triggers the running of `tcpdump` to generate a summary of selectively filtered packets based on relevant IP addresses or ports for a period of five minutes. The resultant `.pcap` file might then be sent to a timestamped directory, and its integrity verified through the generation of a SHA-256 hash, with the results written to an accompanying `.sha256` file.

Later, scripting might also develop the ability to keep logs of activity, run process snapshots (e.g., `ps aux` or `lsof`), and aggregate system metrics, all of which are recorded in addition to the network trace. These files generated through scripting can be made transferable or ready for long-term preservation through archiving and compression, depending on data preservation policies defined in the environment. This entire process can be done in full automated mode through scripting and within defined privilege limitations to ensure the protection and proper processing of sensitive information. To complete the response workflow, Zabbix can be configured to notify an external system such as a central SIEM, a webhook endpoint, or an administrator via email or Telegram, attaching relevant logs or referencing the archive path. This notification closes the loop between detection and operator awareness and enables further inspection by incident handlers.

This architectural system places Zabbix in the role of forensic readiness enabler, ensuring that, in the event of an incident, evidence is always gathered, annotated, hashed, timestamped, and saved in a digital format without any manual intervention required. This system, in one of its numerous lightweight configurations, suits resource-scarce organizations, air-gapped systems, or industrial environments where significant SOAR system installations would be impractical.

The approach adheres to the established theoretical foundations in research articles related to reactive forensics automation [21] and minimal intrusion prepared structures [34]. This demonstrates that common monitoring applications, like Zabbix, can facilitate considerable evidence collection without continuous violations and in compliance with regulatory norms.

3.5 Limitations and Design Considerations

3.5.1 Zabbix Constraints in a Forensic Context

Although Zabbix provides a versatile and extensible model for monitoring networks and systems, its native capabilities were not designed specifically for the scope of digital forensics. This section highlights the main limitations to consider in adapting Zabbix to strengthen effectiveness in forensic analysis.

Table 2.2 shows that Zabbix lacks built-in forensic integrity features; therefore, it requires using ancillary scripts for the handling of evidence. Specifically, Zabbix lacks inbuilt abilities to carry out basic forensic activities, such as full packet capturing, memory acquisition, or disk-image extraction.

Second, the temporal accuracy of the data acquired in Zabbix depends on the internal polling intervals and the synchronization of the clocks of the monitored hosts. In forensic science, timestamps and their granularity play a very important role in reconstructing reliable timelines and proving the randomness of events. If the system clocks of the agents or the Zabbix server are not properly synchronized via NTP, the collected information risks losing forensic validity due to temporal discrepancies. Studies such as [35] indeed underline the importance of having accurate and verifiable timestamps.

Another limiting factor is the processing model for actions and script execution in Zabbix. Scripts triggered through Zabbix actions are run as the user credentials of the Zabbix agent or server and hence inherit environmental restrictions and privileges set up on the host operating system. For example, capturing packets on the network might require root privileges, and configuration errors with respect to such privileges will accidentally hinder the data gathering process. As such, privilege assignments must be carefully set up; *sudoers* policies must be clearly stated; and the script is properly placed to utilize this solution within the context of production efficiently. Additionally, Zabbix lacks built-in functionality for the tamper-evident and secure storing of harvested artifacts. Guarantees for forensic integrity - usually accomplished with the help of hashing algorithms, timestamping, and chain-of-custody documentation - need to be provided through additional scripts or tools. Unlike forensic-focused tools such as *AutoForensics* [21] or Splunk SOAR systems [28], which provide built-in forensic capabilities as shown in Table 2.2, Zabbix requires users to implement these features externally.. As a result, more components are required for the secure archiving of data, such as specially designed servers, append-only logs, or immutable storage-compliant systems.

Although these limitations do not prevent the use of Zabbix for forensic purposes, they require a conscious and competent approach to its use. Administrators need to overcome the lack of built-in forensic functions by using the versatility of Zabbix to import external tools.

3.5.2 Design Philosophy: Separation of Concerns

Despite the admitted limitations, the architectural model is based on a layout to fight complexity and enhance maintainability along the lines of the separation of concerns principle. Each part of the forensic pipeline - namely, the detection, response, acquisition, and preservation - is designed to act autonomously with the possibility of interaction through well-defined interfaces and triggers.

For instance, the detection mechanism built into Zabbix triggers can be individually tuned or optimized separately from the script used for the gathering of evidence. As such, the acquisition module, like a shell script running `tcpdump` or `journalctl`, has the flexibility to replace or upgrade to more advanced tools with minimal need for customization of the detection parameters. Similarly, the preservation module for hashing, timestamping, and the securing of artifacts as archives can individually meet organizational requirements or adapt to regulatory requirements like GDPR or ISO/IEC 27037.

In forensic engineering, this modularity is critical, where sources of evidence, data formats, and legal constraints vary widely across scenarios. A monolithic, strongly coupled design would limit such adaptability and introduce risks at the time when modifications were needed. On the contrary, the system proposed in this thesis allows for versioning, testing, and distributing each component individually, following a low-coupling orchestration approach.

In addition, the definition of logical components contributes to the effectiveness of the safety assessment and compliance with incident response processes. Distributing responsibilities among distinct modules helps define separate boundaries of trust and solidify the principle of least privilege. For example, the script used for hash evaluation can be carried out on a highly constrained system with minimal exposure, while the detection mechanism remains the responsibility of the monitoring staff. Ultimately, employing this modular approach complies with both secure system design principles and good forensic practices. This ensures that the system remains maintainable, extensible, and capable of integrating future improvements into evidence acquisition or preservation technologies without compromising existing detection logic.

3.6 Readiness for Case-Based Applications

3.6.1 Customizability and Scalability of the Model

The working effectiveness of the model of forensic collection is directly related to its ability to respond to new threats as they emerge, adapt to changing infrastructural requirements, and

participate in process refinement exercised continuously. The research-based architecture utilizing Zabbix-enriched automated and modular setups is highly adaptive and scalable for its practical use.

The fundamental reasoning employed for the identification of incidents through the model demonstrates extensive flexibility for differing incident types through the calibration of the parameters being monitored with the corresponding thresholds. For instance, the system can detect attempts to exfiltrate data through the inspection of the amount of outbound network traffic, while privilege escalations might be discovered through abnormal system calls, unintended user group membership changes, or abnormal usages of `sudo`. Such extensions are not required for the strengthening of the facility architecture; instead, they are used to add new triggers involving relevant metrics and activities. Such flexibility is essential for security applications, as the attack strategies continually change.

In addition to that, the acquisition script and response script built into the Zabbix actions model are subject to further refinement and incremental version upgrades. Initial versions might depend on simple tools like `tcpdump` or `ps`; they have the potential to greatly expand by implementing sophisticated data gathering mechanisms, integrating with centralized log storage systems, or providing connectivity to digital forensic storage systems down the road. Since Zabbix is able to take script parameters as input and run any executable available on the server where the Zabbix server program is running, the possible alternatives are truly unlimited. Another form of adaptability is achieved through the ability to adjust triggers by applying historical behavioral baselines. Zabbix is able to apply statistical functions like `avg()`, `count()`, and `trendavg()` across different periods. What this does is allow administrators to set up triggers for finding anomalies based on things other than set values, but on behavior different from the norms normally experienced by the thing being monitored. As such, organizations are able to set up adaptive alarm thresholds, thereby keeping false alarms to a minimum and better reflecting the true operating characteristics of the systems.

The second important criterion is with regard to scalability, more particularly for wide-scale IT or OT deployments. Zabbix originated as software to support distributed monitoring through the use of proxies to allow for horizontal scaling with minimal architectural adjustments. Further agents or proxies can be added seamlessly to handle additional hosts, network sub-blocks, or sensors, all operating within the previously set up trigger-action-collection model. In addition to that, the orchestration logic is not limiting; each trigger is free to trigger its matching actions autonomously and hence support real-time incident identification and processing across different nodes.

The model's scalability and customizability allow it to apply to various settings and threat conditions. No matter if the organization needs to monitor just a few standalone hosts or its entire infrastructure, the modular nature of the system ensures that adaptation and extension of the forensic logic take place with minimal friction over time.

3.6.2 Transition to Real Scenarios

While the model proposed operates on theoretical as well as on structural levels, its empirical effectiveness is subject to verification by systematic testing. In the next chapter of the book, two simulation scenarios for attack were utilized on a virtualized test laboratory to demonstrate the possibility for Zabbix to act as an efficient real-time forensic enabler.

The first situation involves an attempted exfiltration of data by the internal user, who exfiltrates sensitive files to the unauthorized external server. The detection mechanism relies on sudden surges in outbound traffic and anomalies in process forking. As a response, it triggers the logic framework to initiate an acquisition script meant to collect the logs, active network connections, packet captures, and then lock down the former with cryptographic integrity guarantees.

The situation provided is related to the illegal setup of a service, that is, the installation of a hidden web server on a compromised device. In this situation, the detection phase utilizes the Zabbix Low-Level Discovery mechanism to detect hidden services and runs several evidence-gathering procedures involving port scanning, process scanning, and filesystem analysis. In line with the situation above, tamper-resistant integrity is maintained for the harvested artifacts.

The test cases provided not only act as examples of proof-of-concept but are templates for possible implementation in real-world scenarios as well. They illustrate that on-demand invocation is possible for every module - detection, action, acquisition, preservation - without the need for human intervention and hence present relevant forensic data on time. In addition, they highlight the adaptability of the proposed solution for incident-specific requirements as illustrated by the example of minimal redesign for different threat scenarios.

The transition from theoretical design to real-world implementation validates the model as effective and provides insightful angles on the trade-offs and best strategies involved with implementing Zabbix as a forensic automation tool. Due to its simple and flexible architecture, it implies that small to medium-sized businesses with limited resources are able to adopt this approach to enhance their incident response and evidence capture mechanisms despite the lack of investment in enterprise-class SOAR products.

Chapter 4

Workflow Model for Automated Forensic Collection

This chapter provides a full and modular framework for forensic automation of data collection, without any specific monitoring tool. We look for a standard procedure set to be usable with Zabbix or replaceable with any other tools, with the same logical structure and without loss of integrity, traceability, and reproducibility.

We first outline the configuration of the environment, stressing the setup of secure remote access using SSH keys, strict authentication procedures, and automatic verification of required forensic software. Target systems must meet all prerequisites to ensure effective evidence gathering.

We then clarify the evidence collection execution, whereby particular commands are utilized to gather volatile and non-volatile states, as well as process metadata, history of use, and system logs. Each file gathered is immediately hashed utilizing SHA-256 and stored within structured metadata, and consequently, a chain of custody resistant to alterations.

In conclusion, this chapter addresses the topic of secure artifact retrieval and storage, detailing methods for encrypted transmission, archive management, and verification of integrity post-transfer. We conclude with an analysis of the flexibility of these elements, resulting in a dynamic framework that satisfies both enterprise and regulatory standards independently of Zabbix.

4.1 Environment Preparation and Setup

4.1.1 Secure Access and Authentication Framework

The implementation of efficient forensic automation requires a dependable and verifiable methodology for the remote access of specified systems. This framework employs SSH key-based authentication to ensure the integrity and non-repudiation of all operations conducted during the evidence collection procedure.

First, a unique forensic user account is created on each target to be monitored, with permissions of no higher than what's necessary to gather the information. In the master server, the Zabbix server, for example, high-security elliptic-curve or RSA algorithms of a minimum of 4096 bits are employed to create public keys. The corresponding public key is thus stored on the target host, in the forensic user's home directory, to be precise, in `.ssh/authorized_keys`, with a file mode of `600` to guarantee it can not be altered without permission.

Target's SSH server is made secure by disabling password login (`PasswordAuthentication no`) and root login (`PermitRootLogin no`) and enabling only the essential key exchange and cryptography. The directive `AllowUsers` also restricts login to the intended forensic user. Additionally, further utilities like `fail2ban`[36] may be utilized to tightly monitor failed login attempts.

Host-specific `HostKey` lines are also configured on the collection server's `.ssh/known_hosts` file to protect from man-in-the-middle attacks. Connections are set using `StrictHostKeyChecking yes` and `BatchMode yes`, so interactive prompting doesn't occur and your automated scripts do. Pre-collection, a preparatory verification checks SSH access functions, and the forensic user can run necessary commands without prompting for a *sudo* password. A short test script on every target checks necessary binaries (e.g., `ss`, `lsof`, `tcpdump`) and indicates missing dependencies. By enforcing key-based access and a strict SSH policy, this setup allows a safe, tamper-resistant method of remotely controlling every step of the automated forensic workflow.

4.1.2 Prerequisites Verification and Package Installation

Early in the process of collecting on a forensic basis, it's necessary to be assured that the target host has all the pertinent binaries and system utilities to facilitate the successful gathering of evidence. This checking process is done on an automated basis by the collecting script at the time of negotiation for a secure connection.

The script checks for the presence of necessary packages, including `zip` (used for compressing evidence artifacts), `tcpdump` (for gathering network traffic), `psmisc` (`pstree`-based process hierarchy), `net-tools` (for obsolete networking tools like `netstat`), and essential system utilities such as `lsof`, `ss`, `ps`, `sha256sum`, and `df`. In case a necessary package is missing, it informs the operator and automatically attempts to install missing packages (e.g., `apt-get update && apt-get install <missing-packages>` on the scenario of Debian-based systems), then proceeds with its activities.

Under **Case Study I** (detection of data exfiltration - Section 5.2), particular importance is attached to utilizing `tcpdump` and `lsof` for real-time capture of network traffic and maintenance of an active connection under constant surveillance; a lack of them would detract from the completeness of the response. For **Case Study II** (the inappropriate use of services within an industrial setting - Section 5.3), the script also examines permitted access to directories and logging subsystems and checks for Python module availabilities in situations of custom industrial diagnostics being employed.

Automated verification allows for real-time checking of performance, minimizes the chance of partial collections, and facilitates standardization of collections conducted on different systems. In case of necessity, a person can version and collect a customized list of prerequisites suitable for each particular case study, thereby facilitating tool-specific preparations irrespective of the situation.

4.2 Evidence Collection Execution

4.2.1 Focused Data Acquisition and Examination

The preliminary stage of automated forensic acquisition encompasses the methodical gathering of volatile and non-volatile artifacts from all levels of systems. This aspect is engineered with resilience and adaptability as cornerstones to achieve full evidence gathering, irrespective of a system setting or operating condition.

The data acquisition method employed is a unified plan developed particularly for the distinct nature of the target environment. Discovery of network interfaces is done through a range of fallback processes: the script first attempts to find active interfaces using existing utilities (`ip -o link show`), and if this fails to work, it falls back on using standard methods. For each discovered interface, custom packet capture processes are started in parallel, thereby augmenting time coverage and minimizing the time it takes for acquisition.

Its service and process cataloging is performed likewise within an equally rigorous framework to consolidate many data sources and develop a holistic picture of the state of the system. Its workflow correlates active network ports to processes and pulls rich metadata like process IDs, locations of executable files, environmental variables, and open file handles. Binary integrity

checking is accomplished efficiently using SHA-256 hashing of executable files, and file access patterns of process-specific files are acquired using selective `lsof` invocations.

User activity building involves a comprehensive collection of the shell history of all user accounts available. The program goes through the home user directories methodically and discovers and accumulates the shell command histories (bash, zsh), keeping file permission constraints in mind. Administrative accounts are specially taken care of, and efforts are made to restore the root history if permissions allow.

The summarization of system logs allows for sophisticated selective sampling, focusing on log sources considered essential to security (e.g., `auth.log`, `syslog`, and `secure`), and at the same time condensing data size using tail-based extract techniques. The surveillance of recent file changes targets directories marked as high-risk (`/tmp`, `/var/tmp`, and `/opt`), using a time window to be configured to account for recently installed artefacts or settings.

The architecture's context-aware specialization manifests in the industrial setting with additional collection modules aimed at scenario-dependent entities like OT-selective service settings, PLC diagnostic information, and industrial protocol log messages. In this extensible architecture, the cross-platform collection logic is preserved, and the scenario-dependent needs are met with modular additions instead of full redeployment.

4.2.2 Policies on Integrity and Secure Storage

Data integrity preservation is a governing principle throughout the entire process of evidence collection, and it's accomplished using multiple levels of cryptographic verification and access control measures. This multi-level framework ensures the preserved artifacts are maintained with evidential credibility from the beginning of acquisition, long after long-term storage, and on to any eventual legal proceedings.

Instantaneous integrity sealing process proceeds in tandem with the data acquisition process. The first step of being subjected to further operations is to execute a hashing process for all files utilizing the SHA-256 algorithm. The first hashing creates a distinctive marker that is employed within verification processes appearing later. Hashing is conducted on separate components, including process listings, network captures, log segments, and user activity listings, and on aggregated metadata files, thereby establishing an exhaustive integrity model of the complete data collection session.

The subsequent process of verification that confirms information accuracy is known as pre-transfer verification. Before transferring the complete evidence archive across the network, a hash computation is performed to ensure data integrity. This hash serves two primary purposes: it identifies any potential corruption that may occur during the packaging phase and generates a singular value that allows for end-to-end verification throughout the transfer process. The hash is calculated on the receiving end and is recorded in a separate sidecar file that accompanies the evidence archive. Post-transfer validation constitutes the final step of the validation process. The received archive's hash is again recalculated on the collection server and compared with the original hash value provided by the original system. In the event of a mismatch, the user is immediately alerted, and the compromised archive is barred from being entered into the evidentiary storage system. The cleanup of transient files on the target computer takes place only after the hash validation has been completed. This procedure guarantees that the evidence remains retrievable in the event of any transfer errors.

Tamper-evident storage framework works alongside a hashing function, backed by well-defined access control measures. Specific forensic folders are employed to preserve evidence repositories with write-once, read-many access permissions. In the initial process of depositing data, only permitted collection processes have the ability to store data. Afterwards, it is disabled and allows only authorized forensic examiners and audit processes to read the evidence. This ensures that once evidence is incorporated within the storage framework, it can in no way be modified by people who are not authorized, yet still enables required access for conducting ongoing investigations.

4.3 Generalization of Scriptable Logic

The forensic workflow chapter demonstrates a *tool-agnostic* architecture, therefore not dependent on any particular monitoring platform. Even if Zabbix happens to be the main control system here, the fundamental script logic of this solution is intended to be easily adaptable for it to be made compatible with different SIEM systems, SOAR systems, or custom automation deployments, all without jeopardizing forensic processes' integrity and the workflow's efficacy.

Modular abstraction forms the building block of this generalization strategy. Each component of the forensic pipeline - environmental preparation, data acquisition, integrity measurement, and secure storage - represents a separate module with clearly defined input and output specs. The execute model for running remotely, built on SSH, takes standard parameters (target IP, credentials, incident context) and generates predictable output (timestamped archives, hash manifests, execute logs), regardless of the platform on which it was launched. This abstraction allows organizations to deploy the tool in tandem with existing security orchestration solutions like IBM QRadar SOAR, Splunk Phantom, or Microsoft Sentinel without requiring changes to the supporting infrastructure. Standardization of input and output allows for effortless integration of different monitoring systems. Data collection scripts require three major inputs in particular: system identifiers, authorization information, and data relevant to incidents. Output files thus created conform to a standard format, encapsulating a compressed evidence directory, a hash file, and an inclusive report of actions conducted. This standardization allows the framework to work as a forensic microservice within large-scale incident response systems, wherein alerts from across network monitoring utilities (e.g., Nagios and Prometheus), intrusion detection systems (e.g., Suricata and Zeek), or endpoint detection solutions (especially OSSEC and Wazuh) are triggered off runs of a collection process through standardized API calls or command-line invocations. The mechanisms of platform adaptation allow a simple transition to different monitoring solutions. The Zabbix trigger logic, which searches network issues based on thresholds and monitors service discovery, can be replicated on other platforms by their query languages and alert mechanisms. You can, for instance, use Watcher queries in the case of the Elastic Stack in search of the same indicators of compromise, and alert rules in Grafana in order to initiate the same forensic collection processes. The primary collection logic remains the same and requires individual settings of triggers on a particular platform. The framework's extensibility and customizability provide organizations with the means to customize the framework to meet their particular needs without forgoing forensic adequacy. Other collection modules can be incorporated by adhering to a prespecified method: error handling efficiently, maintaining detailed logging, calculating integrity hashes, and chain-of-custody documents are not altered. For example, a few special environments include a requirement for container-related forensics (e.g., collecting Docker or Kubernetes artifacts), evidence acquisition from cloud services such as AWS CloudTrail and Azure Activity Logs, or industrial protocol analysis including Modbus and DNP3 traffic analysis. Each of the extensions possesses the same security standards and legal acceptability as the parent framework. Its vendor-neutral approach enables organizations to implement automated forensic tools without consideration of the security infrastructure currently in place. Such a platform provides a solid base to the automation of incident handling, of the kind characterized by versatility, able to counter technological evolution without losing advanced forensic sophistication levels.

Chapter 5

Implementation and Configuration

Extending substantially on the conceptual theory laid out within Chapter 3, the fourth chapter extends the structure and organization of two simulated attack scenarios placed within a fully controlled virtual lab environment. The overarching task remains to assess the functional efficiency, flexibility, and evidential value of the implemented automated forensic pipeline via Zabbix, while identifying potential practical limitations or challenges likely to arise within the test process.

Both scenarios mimic data exfiltration performed by an insider, resulting in abnormal outbound network activity and the lack of approved file transfers. The second scenario depicts the unauthorized creation of a hidden service, often related to a backdoor listener or an uncommon web server. In both case studies, we begin with a concise threat model that describes the motives of the attacker, environmental limitations, and relevant indicators of compromise. We then define the logical layout of the laboratory, including the Zabbix Server, monitoring agents, the attacking machine, and remote storage, and define how the interaction among these elements enables real-time, holistic detection and automation of evidence collection.

This chapter covers the major configuration parameters, namely trigger definitions, corresponding actions, and evidence storage policies, with an emphasis on major design decisions. Detailed, step-by-step installation and setup procedures for every module are given, along with the full source code for the bash scripts used for collecting the data and hashing, included in Appendix A (*User Manual*) and Appendix B (*Developer Manual*).

This empirical study brings together theoretical concepts and real-world applications to analyze the feasibility and value proposition of the proposed methodology for automated real-time forensics in cybersecurity operations.

5.1 Laboratory Environment Setup

5.1.1 Architecture Overview

The experimental setup is made up of a closed virtual world that contains four different virtual machines, all carefully built to play a specialized role during the collection of forensic evidence. The setup mimics real enterprise networking infrastructures while ensuring total experimental parameter control, minimizing extraneous factors that can disrupt forensic analysis integrity.

At the core of the system is the **Zabbix Server** running on Ubuntu Server 24.04 LTS and Zabbix 7.0 LTS, and is the primary enabler for whole-of-environment monitoring and triggering of forensic activities. The architectural structure of the system, therefore, has a core monitoring MySQL database, a web interface, and alert administration procedures, all working together to enable automated evidence collection procedures. The use of Long Term Support versions enables consistency and stability in the experimental environment over long test periods.

A focused **Victim Host** under Ubuntu 24.04 LTS serves as the main object under study, featuring a running and installed Zabbix agent to send system data and perform remote actions upon certain triggers. The configuration mimics a typical endpoint from a security perspective that is no different from a legitimate entity vulnerable to assaults during a real security incident, involving typical services, including web applications, file transfers, or database work that might be attacked by malicious behaviors.

Network typology employs a hybrid approach that combines NAT and a networking bridge to produce truthful communication schemes, while preserving experimental isolation. This setup allows the laboratory to simulate internal network communications and also external connectivity models, without exposing the test environment to real Internet traffic that could compromise forensic evidence or introduce uncontrolled variables into the experimental design.

5.1.2 Virtual Hosts and Monitoring Targets

In the laboratory setup, a particular IP configuration and a detailed hostname are allocated to every virtual machine, thus facilitating meaningful networking operations and correlating forensic traces between different system components. The schema for addresses is regulated by a well-defined structure that outlines specific roles of different systems in the context of the experimental design framework.

The subnet address range `192.168.20.0/24` was chosen for the network infrastructure, in which the Zabbix Server is `zabbix-srv` (IP: `IP: 192.168.20.120`) and the Victim Host is `victim-host` (`IP: 192.168.20.130`). A correct breakdown of the network is favored by addressing, which clearly differentiates traffic models without damaging functionality for analytical purposes, but also for monitoring activities.

Victim Host setup is implemented to mirror an industrial endpoint by running a range of common services commonly utilized in the enterprise sector. This virtual machine will simulate an industrial appliance, with Modbus and a diagnostic simulator server.

To enhance the likelihood of attack simulations, the Victim Host uses datasets that reproduce sensitive business data, such as personnel lists, financial information, and configuration files containing credentials or system data. The datasets are strategically positioned to create credible data access schemes that, in real situations, would provide forensic evidence for security incidents. This configuration allows detailed testing of the collection of network-facing evidence via multiple file transfer protocols that could be targeted by attackers.

A local NTP server is used in all systems to ensure that system clocks are synchronized, thus ensuring correct correlation of timestamps on collected forensic artifacts. The temporal alignment must be preserved to preserve the integrity of the evidence timelines and support a precise reconstruction of the attack campaigns following the analysis of the incident.

5.1.3 Data Collection and Logging Tools

The evidence collection procedure is built upon a carefully selected set of command-line tools and custom scripts, carefully constructed specifically to gather a variety of evidence classes effectively and maintain integrity during the collection process. These tools work at multiple levels inside the system to allow for all possible evidence sources, from base network traffic to advanced system activities.

Network traffic acquisition is mostly handled by `tcpdump`, a sophisticated command-line packet analysis tool that captures and saves network traffic in PCAP format, all in real-time. The tool is configured to capture all of a packet's content by not truncating, through the use of the *-s 0* flag, and always writes packets out to disk, minimizing temporary data loss during system interruptions, through the use of the -U flag. The acquired network traffic is invaluable when reconstructing attack timelines, identifying data exfiltration threats, and investigating command-and-control traffic.

The system state documentation utilizes a set of readily available Unix tools, including `lsof` to monitor open files and network connections, `ss` and `netstat` to list network sockets, `ps` to inspect process trees, and `top` to monitor system resources in real time. This combination of utilities provides a comprehensive snapshot of system activity at the time of evidence collection, thus helping forensic analysts correlate process behavior with attendant network interactions and file access patterns.

The use of advanced system auditing is made available through `auditd`, which is the Linux Audit Framework. This offers monitoring in kernel space while operating in user space, thus protecting from interference by potentially malicious software. The audit daemon is configured to monitor security-relevant system calls, attempts to access files, privilege elevation, and authentication-related events, creating a comprehensive audit trail that supports real-time detection as well as forensic examination after the fact. The effectiveness of Automation and Orchestration is made possible through the use of custom `bash` scripts that manage workflows related to evidence collection, maintain specified naming conventions, perform cryptographic hashing to verify data integrity, and manage secure storage of collected artifacts. Such scripts include error-handling mechanisms, logging capabilities, and timestamp synchronization, ensuring forensic integrity and reducing risks of human errors in critical operations related to evidence collection.

The dual use of these tools creates a multiplexed system of evidence collection, essentially obtaining ephemeral as well as persistent artifacts while simultaneously maintaining chain-of-custody procedures essential for the legal admissibility of digital forensic evidence.

5.1.4 Security-Oriented Trigger Design

To enable the easy identification of forensic-relevant events with reduced latency and increased accuracy, a set of tailored Zabbix triggers has been created to monitor abnormal behaviors that could indicate a possible compromise. First, *outbound traffic spikes* are identified through the use of a threshold over the `net.if.out` metric. More specifically, we employ the function

$$\text{victim-host:net.if.out[ens160].avg(60)} > 1250000$$

where 125000 represents about 10 Mbps of network traffic. The threshold is a persistent increase in upload volume, and this is a possible sign of a data exfiltration attempt.

Subsequently, unexpected deployment of new services can be monitored via Zabbix's Low-Level Discovery (LLD) on systemd drives. Drives obtained from `systemd.list_services` are compared with a predefined whitelist; any changes or additions to service entries trigger. The detection of unauthorized backdoors that allow privilege escalation is precisely enabled by this described method.

$$\text{victim-host:industrial.port.listen[4444].last()} > 0$$

Every trigger is classified under a *Security* group classification to promote escalation management. The dependencies are organized such that traffic triggers that have a high volume can mute notifications from lower-priority one-socket connections, preventing alert surges. The triggers all have a defined severity macro (for instance `{TRIGGER.SEVERITY}`) to facilitate automated evidence collection procedures, offering information on the needed level of response urgency.

5.1.5 Response Actions and Alerting Mechanisms

Zabbix encapsulates the essential nature of forensic automation by its ability to handle alerts from humans and automated response mechanisms through its robust Actions framework. If a forensically relevant trigger is fired, Zabbix can alert operators while, in the meantime, triggering evidence collection procedures without human intervention.

Hierarchical Alerting and Escalation concept describes various levels of notification channels that mirror incident severity and the corresponding responses required. The escalation system

supports a tiered notification strategy, where primary alerts are sent to first-line technical staff via email, while recurring incidents not resolved within specified time intervals get automatically escalated to more senior officials through SMS or collaborative communication tools [37]. For example, a data exfiltration trigger could immediately alert the SOC team via email, and then notify the CISO after a 30-minute delay in case of no response. Additionally, it could simultaneously alert executive management through integration with Microsoft Teams if the incident persists for more than two hours. Every escalation level operates independently with time thresholds, both for limited escalation sequences and continuous repetition until resolution.

Automatic Response Actions refer to the inherent ability to gather forensic data in real time without the need for operator intervention. Zabbix Actions offer the means to execute remote commands employing several methods: direct system commands via installed agent, SSH-based execution for agentless environments, and custom script execution via the global script mechanism. These automatic responses can be used to initiate packet capture sessions with tcpdump, initiate memory dumps with external tools, or activate extensive evidence collection procedures, all performed within seconds of triggering. The actions framework supports conditional logic, allowing diversified response schemes based on trigger severity, host groups involved, or time criteria.

Multi-Modal Notification Integration goes beyond standard email notifications by including modern collaboration systems and multiple communication channels. Integrated webhook support caters to easy integration with Slack, Microsoft Teams, Telegram, and custom API endpoints, and SMTP configurations support plaintext and HTML messages, injecting forensic context through Zabbix macros. The notification system maintains diligent audit trails, registering all send attempts, delivery checks, and receipt acknowledgments to satisfy forensic chain-of-custody requirements.

Integration of automated procedures and human-initiated alerts facilitates a complete incident response system, whereby technical staff acquire expedient situational awareness, and the collection and storage of forensic evidence occur automatically, ensuring that the capture of perishable evidence proceeds independently from human-initiated delays.

5.1.6 Evidence Collection Scripts

The automated evidence collection process is carried out through a series of modular scripts that can be triggered through Zabbix Actions on either the server or target host. The scripts play a double role: carrying out active mitigation and collecting forensic evidence, and are grouped under three functional groups.

To detect and analyze traffic on specified interfaces for a defined user duration, network capture scripts are used in conjunction with `tcdump`. Scripts can be done remotely via SSH or by the Zabbix agent `system.run` function, accepting parameters related to interface naming, save path, and capture filters, storing files systematically with timestamps.

Momentary data is extracted from system snapshot scripts, also including currently running processes (`ps aux`), available open file descriptors (`lsof`), loaded kernel modules, and current network sockets (`ss`). Snapshots are stored as structured data in JSON format or as simple text files and, consequently, facilitate rapid triage activity and correlation with network acquisitions.

Scripts in the textitIntegrity and Mitigation section create SHA-256 hashes of critical binaries, configuration files, and captured artifacts, while simultaneously recording checksums in metadata logs to ensure the veracity of the chain of custody. In high-severity incident scenarios, scripts can additionally perform active responses, for example, blocking suspicious IP addresses using `iptables` or stopping compromised services via `systemctl stop` commands.

Scripts reside centrally on a repository stored on the Zabbix Server and are delivered securely to your monitored hosts when needed, through automated file distribution mechanisms. Scripts keep a record of their activities, exit codes, and timing information inside a designated Zabbix log file. In this approach, automated forensic processes ensure adaptability, scalability, and sustainability, all while removing the necessity for human interaction on critical incident response activities.

5.2 Case Study I: Detection of Data Exfiltration

5.2.1 Threat Scenario and Indicators

In the first case study, a realistic data exfiltration event is emulated, which covers the unauthorized transfer of about 30 gigabytes of sensitive corporate data from a monitored host system, namely the *victim-host* (IP address: 192.168.20.130), to an attacker-controlled server via the Secure Copy Protocol (SCP).

This attack illustrates a common insider attack pattern, where legitimate authorized credentials and resources were used to access valuable organizational data. The attack uses common network protocols and utilities prevalent in most Unix-based operating systems, thus making it difficult to distinguish between legitimate system administration and malicious activities, especially when there is no proper monitoring in place.

These indicators hint at the likelihood of such an event:

- **Protracted outbound traffic anomalies**: There has been a significant and extended increase in network traffic volume on the victim's primary interface, vastly exceeding baseline patterns set by normal operational behavior;
- **Process execution signatures**: Running of file transfer processes such as `scp`, `rsync`, `curl`, or FTP clients, defined by parameters signifying processes of moving extensive volumes of data;
- **Network Connectivity Patterns**: Opening persistent TCP connections with external hosts, particularly hosts not previously known within the communicative structures of the environment;
- **Resource utilization increases**: There exists a temporal association among increased utilization of CPU, disk I/O, and network interfaces and data transfer activities.

Figures 5.1 and 5.2 show the system under investigation's outbound traffic behaviors and CPU usage over the course of simulating an exfiltration event. The `net.if.out` metrics for interface `ens160` show a dramatic spike from baseline levels near zero to sustained throughput over 600 Mbps, lasting throughout the transfer phase. Although this specific metric cannot be used as a primary detection metric due to its inherently global nature, its correlation with CPU usage trends provides further corroborating evidence of heavy data handling activity.

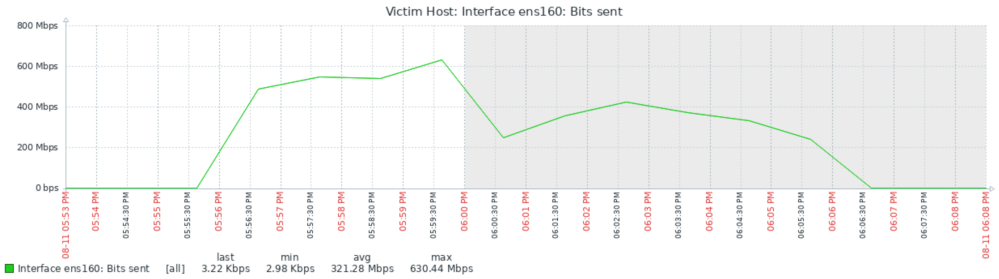

Figure 5.1. Outbound traffic (`net.if.out[ens160]`) measurements showing sustained data exfiltration activity over approximately 9 minutes, with peak throughput exceeding 600 Mbps.

5.2.2 Zabbix Items and Triggers for Network Anomalies

The detection architecture leverages Zabbix's native network discovery capabilities combined with precisely configured monitoring items and trigger logic to achieve rapid anomaly identification with minimal false positive rates.

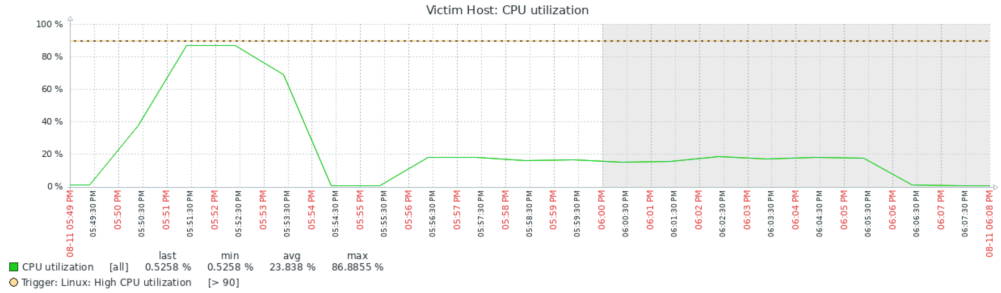

Figure 5.2. CPU utilization correlation during the data transfer window, demonstrating system resource consumption associated with large-scale file operations.

Network Interface Discovery and Monitoring

Before the adoption of targeted traffic tracking, the victim host was configured using Zabbix's Low-Level Discovery (LLD) functionality, which let the system automatically find and monitor all available network interfaces. The discovery service identified the primary interface `ens160`, instead of the more traditionally anticipated `eth0`, thus ensuring that the tracking system properly accommodated the network settings of the target system.

The autodiscovery protocol employed the property `net.if.discovery`, which produces a JSON array containing all network interfaces and their related attributes:

```
{#IFNAME} = "ens160"
{#IFDESCR} = "ens160"
{#IFTYPE} = "6"
```

For each interface found to be meeting operational parameters (status *up* and not *loopback*), the following monitoring components were automatically created:

- `net.if.in[ens160]` – volume of input data in bytes per second
- `net.if.out[ens160]` – outbound traffic in bytes per second
- `net.if.total[ens160]` – interface cumulative

Other elements of network monitoring included:

- `net.tcp.listen[10050]` – check the availability of the Zabbix agent.
- `net.if.collisions[ens160]` – network error identification
- Custom user parameters executing `ss -s` and `netstat -s` for connection state enumeration

Trigger Configuration and Logic

The primary detection mechanism was defined by carefully determined threshold values based on the baseline traffic conditions of the environment being monitored. The trigger expression was given as:

```
{victim-host:net.if.out[ens160].avg(60)}>1250000
```

This alert occurs when the average data transfer rate of outward data exceeds 1,250,000 bytes per second (about 10 Mbps) during a 60-second interval. The level was deliberately set greater than ordinary admin data traffic but below levels typical of common backup processes, thus balancing the sensitivity of detection and realistic operational utility.

The system has an improved recovery system meant to prevent oscillation under margin situations:

```
victim-host:net.if.out[ens160].avg(1800)}<1250000
```

The recovery criterion requires traffic to persist below the specified threshold without interruption for 30 minutes (1800 seconds) before the clearing of the alert condition to avoid intermittent disruptions of data transmission, restarting the incident response procedure. The level of the trigger was set as *High* to enable fast escalation via the alerting hierarchy of the monitoring system. Additionally, auxiliary dependencies were introduced to prevent the triggering of alerts during scheduled times of system maintenance or when the Zabbix agent cannot be reached.

5.2.3 SSH Infrastructure and Authentication Framework

The automated evidence-gathering system requires a secure, password-free authentication method between the clients being monitored and the Zabbix Server (192.168.20.120). This was accomplished using RSA public key cryptography and separate security hardening mechanisms.

Creation and Installation of SSH Keys

In this forensic collection service, a key pair using RSA was created as defined on the Zabbix Server.

```
ssh-keygen -t rsa
```

The resulting public key was added to the approved keys configuration of each monitored host:

```
ssh-copy-id forensic@192.168.20.130
```

To support security controls, the SSH server setup on the monitored systems was limited to only accepting key-based authentication for forensic data collection purposes, along with additional restrictions that limited the key's use to specific command execution patterns.

SSH Session Verification and Protection

All SSH connections made by the evidence collection system employ strict host key verification and cautious parameterization of connection options:

```
ssh -i /etc/zabbix/.ssh/forensic_key -o StrictHostKeyChecking=yes
-o ConnectTimeout=30 -o BatchMode=yes forensic@endpoint command
```

Connection attempts are logged on either end to create an extensive audit trail of forensic access exercises. Inclusion of the `BatchMode=yes` parameter ensures interactive prompts won't compromise the productivity of the automated data gathering procedure, and `ConnectTimeout=30` helps to prevent delays related to unresponsive targets.

5.2.4 Client System Preparation and Dependencies

Successful evidence gathering requires monitored systems to maintain specific software packages and configurations. A comprehensive dependency verification process was implemented to ensure the reliability of collection under changing system configurations.

5.2.5 Package Installation Requirement

The script intended for evidence collection performs automatic dependency verification for the following critical packages, which might not be provided with the guest operating system:

- **zip/unzip**: Essential to creating evidence archives and verifying their integrity.
- **tcpdump**: Network packet capturing tool that requires high privileges.
- **psmisc**: Process hierarchy tools including `pstree` for relationship mapping
- **net-tools**: Core networking tools like `netstat` to guarantee compatibility.
- **lsof**: Listing of open files and network sockets

On Ubuntu/Debian systems, these dependencies are installed via:

```
apt-get update && apt-get install -y zip tcpdump psmisc net-tools lsof
```

The collection script checks if packages are available during the time of initialization and prints useful warnings if necessary tools are missing, thereby achieving progressive degradation of the collections rather than complete failure.

Privilege and Sudo Configuration Criteria

Several of the previous evidence acquisition methods require higher permissions, particularly network packet capture with `tcpdump` and thorough file system analysis using `lsof`. Rather than constantly asking for root access to the account used for acquisition, another sudo configuration was set up:

```
forensic ALL=(ALL) NOPASSWD: /usr/sbin/tcpdump, /usr/bin/lsof
```

This setting allows the forensic collection account to run certain privileged commands without requiring password authentication, allowing for automated processes while maintaining security boundaries.

Automated Logging and Packet Capture Activation

With the triggering activation, Zabbix initiates a full evidence collection process through its *Action mechanism*, running a centralized forensic script on the Zabbix Server. This design ensures consistency in collection methods for all monitored hosts, while at the same time maintaining centralized management and logging.

Implementation Roadmap

The Zabbix Action named *Data Loss Prevention - Forensic Collection* was created with the following operational parameters:

- **Conditions**: Trigger = *Data Loss Prevention Alert (>10Mbps for 1m)*
  AND Trigger value = *PROBLEM*
- **Operations**: Remote command execution with 30-second timeout
- **Recovery Operations**: Notification to the incident response team with the evidence archive location

The Action starts the forensic collection script with dynamically populated parameters, based on the configuration of the triggering host:

```
/usr/local/bin/collect_remote_evidence.sh {HOST.IP} {HOST.FORENSICUSERNAME}
```

Here, `HOST.IP` represents the Internet Protocol address of the first host, and the latter `HOST.FORENSICUSERNAME` refers to the account under which Secure Shell (SSH) authentication is carried out.

Workflow for Evidence Collection and Data Acquisition

Evidence-gathering procedure solely takes place on the intended host via SSH remote command executions and strictly follows an accurately defined order:

1. **Environment Preparation**
   - Creation of a timestamped directory with appropriate permissions for operations (e.g. `/tmp/forensics_192.168.20.130_20250812_133515`)
2. **Volatile Data Capture**
   - Running process enumeration via `ps aux`, capturing PID, PPID, command lines, execution time, and resource utilization
   - Hierarchical process relationships through `ps -eo pid,ppid,cmd,etime,user` and/or `pstree -p` command
   - Network socket tables via `ss -tunap` and `netstat -anp` for established connections
   - Open file descriptor enumeration through `lsof`, identifying all files, network connections, and system resources in use
3. **Historical Data Acquisition**
   - History of shell commands typed everywhere (e.g., `.bash_history`, `.zsh_history`)
   - System log segments from `syslog`, `auth.log`, `messages`
   - Authentication and access logs for correlation with network events
4. **Network Traffic Acquistition**
   - Issuing the command `ip -o link show` allows active network interfaces to be identified automatically
   - Concurrent execution of `tcpdump` on each interface with 60-second capture duration
   - Complete packet acquisition (`-s 0`) and prompt clearing of the buffer (`-U`) to ensure data integrity
   - Individual PCAP files per interface help to isolate information and improve the analytical processes
5. **Metadata Generation**
   - Determining hosts (hostname, IP addresses, and system information)
   - Collection of timestamps and execution context
   - Network interface configurations and status
   - Complete inventory of collected files with sizes and hashes
6. **Cryptographic Integrity Protections**
   - Individual file hashes stored in `hashes_20250812_133515.sha256`
   - Archive-level hash verification after compression

- Hash file formatting that is verification compatible using `sha256sum -c`

7. **Transfer and Compression**
   - ZIP archive creation containing all the materials collected
   - Archive hash calculation using only filenames without including absolute paths
   - Verified safe transmission using SCP to the Zabbix Server evidence repository

Evidence Artifacts and Data Analysis

The applied collection process for this case study yielded the development of a large evidence repository containing the artifacts listed in Table 5.1. The collection comprised a total of 10 main evidence files with their associated metadata, ranging from brief network state dumps to large open file lists.

The full forensic implications were obtained from

- **Network connection data** reveals a substantial number of `TIME_WAIT` connections from the exploited host to the Zabbix server, implying a period of heightened monitoring activity during the incident;
- **Structured data about processes** unveils the occurrence of established SSH connections, namely, (`sshd: forensic@pts/0` and `sshd: forensic@notty`), corresponding to the evidence collection session itself;
- **Command history analysis** for all user accounts, including system installation steps, like the installation of packages and setting up SSH keys;
- **Network interface configuration** confirming the primary interface designation as `ens160` rather than conventional `eth0` naming.

Figure 5.3 shows the state of the Zabbix dashboard at the moment of the trigger activation, highlighting the recording of the issue event and the launch of the automated response procedure.

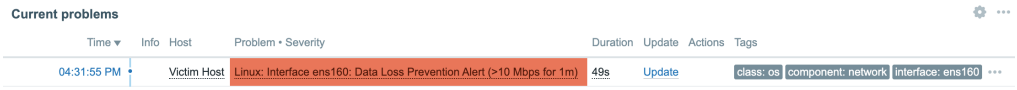

Figure 5.3. Zabbix monitoring dashboard at the moment of trigger activation, showing the high outbound traffic problem event and automated response initiation.

5.2.6 Execution Summary

The complete detection-to-evidence-collection workflow operated autonomously without human intervention, demonstrating the viability of fully automated forensic readiness in network security monitoring environments. From initial trigger activation at 13:35:15 UTC on August 12, 2025, to completion of evidence archive transfer and integrity verification, the entire process was completed within approximately 90 seconds.

The collected evidence dataset provides a comprehensive snapshot of system state during the simulated data exfiltration event, enabling detailed forensic analysis of:

- Process execution timelines correlating with network traffic anomalies;
- Network connection establishment patterns and external communication attempts;
- System resource utilization during large-scale data transfer operations;

- Historical command execution patterns indicating preparation or reconnaissance activities.

The successful execution validates both the technical implementation of the automated collection framework and the operational procedures for evidence preservation and chain of custody maintenance. The cryptographic integrity protections ensure that collected evidence meets forensic standards for admissibility and reliability.

This foundational evidence dataset serves as the basis for detailed analytical procedures documented in Chapter 5, where correlation analysis, timeline reconstruction, and threat attribution methodologies are applied to demonstrate the investigative value of automated collection systems in network security incidents.

Table 5.1. Forensic artifacts collected during Case Study I execution

Artifact Filename	**Size**	**Content Description**
processes_20250812_133515.txt	19386	Complete process listing (`ps aux`)
open_files_20250812_133515.txt	418635	Global file descriptor table (`lsof`)
process_hierarchy_20250812_133515.txt	1776	Process tree structure (`pstree -p`)
process_tree_20250812_133515.txt	13550	Detailed process relationships
netstat_20250812_133515.txt	17744	Network conn. tables (`netstat`)
sockets_20250812_133515.txt	25602	Socket state info (`ss -tunap`)
network_files_20250812_133515.txt	3150	Network file descriptors (`lsof -i`)
collection_metadata_20250812_133515.txt	1728	Host metadata and collection invent.
root_bash_history_20250812_133515.txt	1393	Root account command history
pier_bash_history_20250812_133515.txt	1915	User account command history
hashes_20250812_133515.sha256	790	SHA-256 integrity checksums
packets_ens160_20250812_133515.pcap	17664639	Network packet capture (60 seconds)
Total Evidence Files	**12**	**Complete system state snapshot**

5.3 Case Study II: Unauthorized Service Deployment

5.3.1 Threat Scenario and Indicators

The second case study analyzes a sophisticated attack strategy commonly seen in industrial and operational technology environments: the unauthorized deployment of network services intended to provide constant access while pretending to be legitimate industrial diagnostic interfaces. Such an attack is a severe security flaw in environments where stringent service baselines are critical, notably in industrial control systems, manufacturing networks, and segregated operational environments. Any slight deviation from approved configurations in these environments can be a sign of activity associated with advanced persistent threats.

The attack pattern depicts a case where an adversary has gained initial access to a closely monitored host system in a commercial environment and thereupon strives to set up lasting access by deploying a bespoke service written in Python, which mimics legitimate industrial diagnostic sessions. This malicious service sets up a connection on TCP port 4444 and combines messaging and functionality with a theme of industrial procedures, minimizing the risk of rapid detection during typical penetration testing or ordinary administrative activity.

This attack pattern demonstrates the methods used by attackers in practical situations focused on critical infrastructure, where attackers utilize the complexity and variety of industrial environments to hide malicious services within legitimate operational technology resources. Modern industrial networks almost always support a variety of diagnostic interfaces, protocol gateways, and monitoring services on non-standard ports, thus creating a situation where unauthorized services can coexist undetected for extended periods in the lack of proper monitoring provisions.

The threat landscape covers various attack vectors, which are playing increasingly important parts in contemporary cybersecurity events:

Backdoor Deployment: One of the basic tactics is building lasting remote access by employing lightweight network listeners, thus allowing for command execution with negligible disruption of system usage. Such services are carefully crafted to mimic legitimate industrial procedures, including proper interface definitions and supporting status report facilities.

Impersonation of Industrial Protocols: Advanced attackers are increasingly employing tactics that mimic real industrial communication protocols and diagnostic interfaces, making detection challenging unless proper protocol analysis or proper service baseline monitoring has taken place. This is a hidden service that produces ostensibly legitimate replies to industrial status queries and maintains predictable messaging patterns that conform to the expected behavior of industrial interfaces.

Stealth Characteristics of Operations: The illegal service uses sophisticated evasion techniques, including minimal use of resources, unusual installation mechanisms, and adaptable port usage, all designed to help it avoid detection by regular security monitoring systems. This service has an essentially low system footprint, which allows it to operate below detection levels set by normal endpoint protection systems.

Critical indicators of compromise (IoC) tied to the illegal use of services go beyond basic network monitoring and include features relevant to both behavioral and contextual analysis:

- **Unrecognized Network Listening Ports**: Detection of unrecognized network services, particularly those associated with ports which are not part of the organization's service whitelisting or defined within established network architecture guidelines.
- **Detection of Anomalous Process Execution Patterns**: Detection of interpreter processes, such as Python, Ruby, or Node.js, to run scripts from temporary directories or unexpected alternate file system locations, generally associated with transitory or unauthorized activities.
- **Patterns of Network Connectivity**: Identifying dormant ports handling inbound connection requests from outside hosts or questionable IP addresses can be a potential indicator of reconnaissance activity or the establishment of command-and-control activity.
- **Service Registration Anomalies**: The discovery of services running outside of defined system management silos, including systemd, points to the possible presence of unauthorized installs that bypass traditional configuration methods and installation procedures.
- **Process Hierarchy Irregularities**: The identification of network services started by user shells or interactive sessions, rather than processes started by system initialization, suggests the likelihood of manual deployment rather than approved service installation.

The attack simulator combines realistic operational scenarios with environmental constraints to ensure detection mechanisms are tested within scenarios that are close representations of real-world deployments within industrial networks. Within the test laboratory, real industrial services utilizing default ports (502 for Modbus TCP and 8080 for industrial Web applications) are used to set up natural baseline scenarios and to test the detection system's ability to differentiate between legitimate and illegitimate network services.

5.3.2 Using Zabbix Low-Level Discovery for Service Monitoring

The unauthorized service deployment identification framework utilizes the Low-Level Discovery (LLD) functionality of Zabbix to achieve a comprehensive and dynamic insight into network service levels within the monitored infrastructure. This approach enables frequent observation of service deployment operations, together with a comprehension of the inherently dynamic nature of modern operational technology systems, where services can be appropriately integrated, altered, or removed within typical operational procedures.

Discovery Rule Configuration and Architecture

The basic detection functionality employs a sophisticated, customized discovery protocol, which systematically documents active network service resources by correlating multiple complementary data feeds. This system aims to ensure complete network resource identification, thus minimizing service detection evasion:

```
Discovery Rule: Industrial Services Discovery
Key: industrial.services.discovery
Update Interval: 2 minutes
Keep Lost Resources: 60 days
Filter: {#PROCESS} matches @(python3|python|node|ruby)
```

The discovery script combines the output of several system utilities to generate a comprehensive list of network services, utilizing various data-gathering methods to ensure the accuracy of detection:

Socket State Analysis: The first enumeration methodology utilizes the `ss -tulpn` command to list all currently active listening TCP and UDP sockets and their respective process information. This methodology is established as the most reliable method of discovery of active network listeners, regardless of their particular implementation procedures or service registration requirements.

Cross-Referencing Network Statistics: The `netstat -anp` command enables one to retrieve additional information on connection statuses and corresponding processes, which can help verify socket state information obtained through the primary enumeration method and aid in the discovery of potential mechanisms of service hiding.

Process Validation and Correlation: The cross-check of listening procedures next to output from `lsof -i` enables clear process identification and enables detection of sophisticated methods of process hiding or impersonation, which can evade simpler enumeration approaches.

The discovery script generates Low-Level Discovery information in JSON, including detailed metadata about each service discovered:

```
{
  "data": [
    {
      "{#PORT}": "4444",
      "{#PID}": "2043",
      "{#PROCESS}": "python3",
      "{#CMDLINE}": "/tmp/stealthy_backdoor.py",
      "{#USER}": "bob"
    },
    {
      "{#PORT}": "502",
      "{#PID}": "1847",
      "{#PROCESS}": "python3",
      "{#CMDLINE}": "/opt/modbus/modbus_server.py",
      "{#USER}": "modbus"
    }
  ]
}
```

enabling the automatic creation of monitoring elements for every service discovered, and also providing adequate context information required to execute actions related to forensic investigations and threat attribution.

Item Prototype Configuration and Monitoring Logic

The system develops an independent monitoring component for every service discovered by the Low-Level Discovery process, to monitor the operational status and behavioral properties of the discovered service:

Port Listening Status Monitoring:

```
Name: Industrial Port {#PORT} Listening Status
Key: industrial.port.listen[{#PORT}]
Type: Zabbix agent
Update interval: 30 seconds
History storage period: 7 days
```

Process Existence Verification:

```
Name: Industrial Service Process {#PROCESS} on Port {#PORT}
Key: industrial.process.check[{#CMDLINE}]
Type: Zabbix agent
Update interval: 1 minute
History storage period: 30 days
```

Service Connectivity Testing:

```
Name: Industrial Port {#PORT} Connectivity Check
Key: industrial.modbus.check[127.0.0.1,{#PORT}]
Type: Zabbix agent
Update interval: 2 minutes
History storage period: 7 days
```

The design of the prototype enables exhaustive investigation in relation to appropriate data retention approaches, which balance forensic requirements with storage optimality concerns.

Discovery Script Implementation and Logic

The discovery script of industrial service utilizes advanced enumeration tactics specifically crafted to work within the complex and diverse dynamics of operational technology systems. It works by systematically scanning a fixed set of ports usually associated with industrial protocols, such as legacy Modbus TCP (502), industrial Web portals (8080), MQTT communication ports (1883/8883), and countless proprietary industrial protocols (20000, 44818, 47808, 102). This set of ports reflects legitimate industrial network architectures, where a diverse set of customized protocols work in conjunction with one another to enable the functionality of systems such as manufacturing, process control, and facilities management.

The enumeration process employs different system utilities in conjunction to gather detailed data regarding services. The script calls the `ss` command under specific parameters to identify network sockets in the listening state, then performs a detailed process analysis to extract relevant contextual data related to each identified service. For every active port discovered, the discovery process performs process identification through PID extraction, command-line analysis, and user context assessment, thus providing detailed information regarding service characterization.

The script uses effective error-handling mechanisms, which ensure the continuation of functionality even in the event of failure or incompleteness of individual process queries. Where process information is unavailable, fallback values are used, which reliably prevent script failure and ensure continuity of the discovery process. Additionally, the use of defensive programming mechanisms enables effective functionality on a variety of system configurations and variable applications of security policy, which would otherwise obscure process visibility. The output format conforms to the strict JSON Low-Level Discovery specifications defined by the Zabbix discovery

engine. Therefore, the script generates properly formatted JSON output containing definitions of macros like PORT, PID, PROCESS, CMDLINE, and USER, thus enabling the automated setup of monitoring items enriched with contextual information. The JSON output is validated for format using the `jq` tool to ensure both syntactical correctness and conformity with Zabbix's parsing requirements.

The discovery process utilizes a sophisticated port-scanning technique aimed at reducing unnecessary resource usage while at the same time confirming accurate identification of relevant industrial services. Rather than performing sweeping network scans, the script focuses on ports known to be associated with established industrial protocols, relieving system load and minimizing the chances of interfering with operational technology systems vulnerable to network interaction.

The data-validation mechanisms built into the script ensure that the data collected on processes accurately reflects the system state, thus preventing propagation of false information throughout the monitoring system. Process presence analysis, command-line input validation, and user context investigation offer further protection for data integrity, thus enhancing the reliability of the follow-on monitoring and notification operations associated with known services.

5.3.3 Trigger Logic and Detection Algorithms

The detection technique utilizes a complex, multi-level approach consisting of baseline comparison, anomaly detection, and context validation. This technique exhibits a low false positive incidence and is effective in tracing services that are indeed unauthorized.

Primary Detection Trigger Configuration

The core detection principle utilizes the ability of Zabbix to trigger on newly detected services that are above the threshold of approved industrial services:

```
Trigger Name: Unauthorized Industrial Service Detected on Port {#PORT}
Expression: {industrial-host:industrial.port.listen[{#PORT}].change()}>0
            and {industrial-host:industrial.port.listen[{#PORT}].last()}>0
Severity: High
Recovery Expression: {industrial-host:industrial.port.listen[{#PORT}].last()}=0
```

This trigger setup includes multiple levels of verification to ensure detection accuracy:

The `change()` method detects the time when a previously non-monitored port starts to actively listen, which means a new available network service.

The `last()` function checks that the service in question still actively listens, thus avoiding false positives caused by transient network conditions or temporary port bindings.

String pattern analysis of command line histories can be useful in identifying potentially malicious service implementations.

Advanced Trigger Logic and Correlation

Then, the detection system employs sophisticated correlation logic, which identifies legitimate patterns of industrial service usage and maintains its reactivity to wrongful usage:

```
Trigger Name: Suspicious Process Execution Pattern - Industrial Network
Expression: ({industrial-host:industrial.process.check[python].last()}>0
            and {industrial-host:industrial.port.listen[4444].last()}>0)
Severity: Warning
Dependencies: Network Infrastructure Maintenance Window
```

The correlation trigger also evaluates multiple system properties:

Process-Port Correlation: Detection logic identifies when interpreter processes (Python, Ruby, Node.js) are associated with network listeners on non-standard ports, indicating potential unauthorized script execution.

Host Context Validation: This process combines host identification protocols intended to distinguish between systems authorized to run diagnostic services and systems where such a service would be a violation of security.

Maintenance Window Integration: Dependency relationships prevent false alarms during scheduled maintenance activities when legitimate diagnostic tools may be temporarily deployed.

Baseline Management and Whitelist Implementation

The detection process utilizes adaptive service baselines, which include sanctioned operating adjustments and the detection of unauthorized changes:

```
# Approved Industrial Services Configuration
Macro: {$APPROVED_PORTS}
Value: 502,8080,1883,8883,20000
Description: Whitelisted ports for legitimate industrial services

Macro: {$APPROVED_PROCESSES}
Value: modbus_simulator.py,diagnostic_server.py,scada_interface
Description: Authorized industrial service executables
```

Baseline's validation logic integrates the above macros in the trigger expressions:

```
Trigger Condition: {industrial-host:industrial.port.listen[{#PORT}].last()}>0
                   and {#PORT} not in {$APPROVED_PORTS}
                   and {#PROCESS} not in {$APPROVED_PROCESSES}
```

This approach supports service authorization management while, at the same time, providing automatic detection of unauthorized implementations.

5.3.4 Scripted Response and Evidence Collection

When an unauthorized service deployment is discovered, the Zabbix action system triggers an in-depth protocol for evidence collection by running a custom industrial forensic collection script. This evidence collection process employs an advanced forensic acquisition framework that has been carefully crafted for OT environments, where conventional forensic practices may be ineffective or inappropriate because of operational limitations and the fragility of the involved systems.

Action Configuration and Trigger Response

The Zabbix Action process is outlined with clear parameters to ensure proper initiation of evidence collection:

```
Action Name: Industrial Security - Unauthorized Service Response
Conditions:
  - Trigger = "Unauthorized Industrial Service Detected on Port {#PORT}"
  - Trigger value = PROBLEM
  - Host group = Industrial Systems
Operations:
  - Remote command: /usr/local/bin/industrial.sh {HOST.IP} {ITEM.KEY} forensic
```

```
  - Command timeout: 180 seconds
  - Execute on: Zabbix server
Recovery Operations:
  - Send notification: Security team with evidence archive location
```

The operation setup involves dynamic parameter transmission, which provides key contextual information about the incident that triggered the action and thus enables precise and effective gathering of information related to the particular security event.

Evidence Collection Workflow and Methodology

The forensic collection process adopts a systematic methodology that captures significant information about the current state of the system, focusing on traces relevant to unauthorized service installations.

1. **Environment Preparation and Initialization**
   - Creation of timestamped evidence directory with secure permissions: `/tmp/forensics_{TARGET_HOST}_{TIMESTAMP}`
   - Initialization of collection metadata, including session identifiers, timestamps, and tool versions information
   - Validation of collection tool availability and privilege requirements
2. **Network Service Enumeration and Analysis**
   - Complete port scanning and listener identification using `ss -tulpn` for comprehensive network service inventory
   - Cross-correlation with `netstat -anp` output to ensure detection completeness and identify potential evasion attempts
   - Inspecting network file descriptors using `lsof -i` to capture network activity that goes beyond simple port listening
3. **Process Investigation and Binary Analysis**
   - Comprehensive list of processes, including parent-child relationships, command-line arguments, and the execution context
   - Detection of binary paths and generation of cryptographic hashes towards executable integrity verification
   - Obtaining environmental parameters relevant to every procedure associated with network listeners
   - Memory mapping analysis for enhancement of advanced malware detection capabilities
4. **Historical Activity Reconstruction**
   - Command history collection from all user accounts, including root and service accounts
   - System logs analysis focusing on authentication events, process executions, and network activity
   - Examination of the file system timeline related to recently modified files within temporary and application directories
5. **Industrial-Specific Artifact Collection**
   - Analysis of industrial application directories (`/opt/industrial/`) for configuration files and log data
   - Collection of protocol-specific logs and communication traces relevant to industrial operations

- Valid industrial services record for foundational comparison and attack attribution

6. **System State Documentation**
   - Running service enumeration through `systemctl list-units` for complete service inventory
   - Resource utilization metrics, which include CPU, memory, and disk usage during the incident period
   - Configuring network interfaces and checking the routing table
   - Security policy and access control mechanism verification

Cryptographic Integrity and Chain of Custody

The evidence collection procedure includes rigorous integrity protection measures intended to ensure forensic admissibility:

Individual Artifact Hashing: Each collected file undergoes immediate SHA-256 hash calculation upon creation, with hash values stored in corresponding *.sha256* files for independent verification;

Consolidated Hash Repository: Distinct hash values are combined into a large forensic report, thus making central integrity verification and detection of potential tampering easier;

Archive-Level Verification: The overall evidence archive is subjected to cryptographic hash computation before and after transfer, hence allowing detection of corruption or tampering in case they happen during collection or transfer procedures;

Temporal Attestation: During the collection, timestamp information is incorporated utilizing system clocks set by NTP, allowing timelines to be reconstructed accurately and correlated with outside event sources;

Chain of Custody Documentation: Detailed documentation on collection methods, use of instruments, and evidence management must be maintained to ensure compliance with legalities related to digital forensic evidence.

5.3.5 Laboratory Implementation and Testing

Test Environment Configuration and Baseline Establishment

The evaluation of the unauthorized use of services was carried out in the given study case environment described in Section 5.1, which utilized an enhanced network topology and a monitoring system designed to mimic industrial systems.

The implementation of this evaluation required careful cooperation between the legitimate industrial services and the malicious backdoor service to generate realistic detection cases accurately reflective of operational technology configurations.

Reliable operating parameters were determined by the use of two main industrial services, which provided realistic background conditions for the detection system:

Modbus TCP Simulator Service:

```
Service: /opt/industrial/modbus_simulator.py
Port: 502 (Standard Modbus TCP)
Process: python3
User: modbus
Description: Legitimate industrial protocol simulator
```

Industrial Diagnostic Web Interface:

```
Service: /opt/industrial/diagnostic_server.py
Port: 8080 (Industrial Web Interface)
Process: python3
User: apache
Description: Authorized diagnostic and monitoring interface
```

The above legitimate services were specifically created using Python interpreters in order to test the detection system's ability to distinguish between authorized and unauthorized Python-based services running in a common environment, thus testing the effectiveness of the monitoring framework's context analysis.

Unauthorized Service Implementation and Deployment

The backdoor service simulator utilizes an advanced implementation that utilizes Python to emulate the behavioral patterns and interface functionalities characteristic of authentic industrial diagnostic systems within operational technology environments. The service is crafted with malicious purpose and runs on TCP port 4444, taking advantage of complex masquerading mechanisms to ensure smooth integration into legitimate industrial monitoring systems, thus allowing for stealthy remote access by simulated intruders.

The architecture design centers on a multi-threaded network service with the ability to efficiently manage multiple client connections at one time, all the while maintaining the operating look and feel of a standard industrial diagnostic interface. Service start-up entails checking available ports in order to avoid potential conflicts with currently active network services, followed by the typical TCP-socket binding and listener setup usually seen in industrial service deployment. The threading design ensures individual client connections do not hinder the fundamental workings of the core service, thus allowing for the real-time access characteristic of true industrial monitoring systems.

The backdoor service has sophisticated capabilities that enable it to mimic industrial protocols, hence making it possible to offer responses to basic diagnostic questions in data formats and response types indistinguishable from actual conditions. When status questions regarding the system are asked, the service generates realistic responses that include emulated data from programmable logic controllers, readings from environmental sensors like temperature and pressure, and performance indicators like production rates and machine utilization statistics. The responses are carefully crafted to follow the formats and value ranges common in legitimate industrial diagnostic interfaces, making it difficult to detect using casual manual examination.

Masquerading itself as a legitimate operational infrastructure, the service integrates a broad command execution feature that provides remote attackers complete access to system functionalities. This command execution feature operates through a hidden interface that captures any command of the system and sends its output over the defined network channel. Employing its dual-mode capacity, the service can sustain the illusion of complete legitimacy during normal system administrative operations while at the same time providing full system takeover via the hidden command interface.

The service utilizes a range of covert operating components constructed to fall below the threshold of detection of typical security surveillance systems. Patterns of resource utilization are carefully aligned with the typical parameters associated with legitimate industrial activity, thus evading detection systems invoked by increased CPU or memory utilization which would invoke resource-based detection systems of anomalies. Additionally, network traffic patterns are carefully monitored to avoid developing unusual communication patterns, with connection management tailored to emulate the irregular connectivity pattern associated with legitimate diagnostic interfaces.

The error management mechanisms and operational robustness of the backdoor service ensure its reliable availability, even under adverse system conditions. Conflict port detection at

deployment minimizes the likelihood of service disruption, while sufficient error-recovery mechanisms provide continued functionality on occurrence of network outages or shortages of system resources. Additionally, the service includes logging-suppression mechanisms that minimize the production of forensic information and maintain the reporting of operational status typical of a legitimate industrial diagnostic tool. The effort aims at refining automated systems' detection ability by integrating terms applicable in business scenarios, thus producing status reports highly analogous to real documents. Besides, it embraces operational behaviors emulating those in actual industrial diagnostic procedures. This all-inclusive simulation tactic evaluates the detection system's ability to detect complex threats utilizing sophisticated evasive tactics, instead of relying on conspicuous malignant signatures or behavioral patterns easily discerned by conventional security surveillance procedures.

Zabbix Agent Configuration for Industrial Monitoring

The client host requires a customized agent configuration to support Low-Level Discovery (LLD) for industrial service monitoring:

```
# /etc/zabbix/zabbix_agentd.conf additions
UserParameter=industrial.services.discovery,/usr/local/bin/ind_discovery.sh
UserParameter=industrial.process.check[*],pgrep -f "$1" | wc -l
UserParameter=industrial.port.listen[*],ss -tlnp | grep -c ":$1 "
```

Additionally, the forensic collection competency requires additional permissions for certain actions:

```
# /etc/sudoers.d/forensic-collection
forensic ALL=(ALL) NOPASSWD:/usr/sbin/tcpdump /usr/bin/lsof /bin/ss /bin/netstat
```

This configuration allows automated evidence collection with the minimum privilege needed for the collection, thus not violating system integrity.

Detection Verification and Timing Analysis

The Zabbix monitoring system rapidly detected the unauthorized use of services within the specified discovery period. Careful examination of the timing demonstrates the complete timeline of both detection and response:

```
Timeline Analysis - Unauthorized Service Detection
================================================
T+00:00  Backdoor service deployment initiated on victim host
T+01:45  Scheduled LLD execution discovers new service on port 4444
T+01:47  Item prototypes execution for newly discovered service
T+01:52  Initial data collection confirms persistent service operation
T+02:05  Primary detection trigger evaluates and fires (unauthorized port)
T+02:08  Zabbix Action framework initiates forensic collection script
T+02:12  SSH connection established and evidence collection starts
T+03:28  Collection process completes and evidence archive created
T+03:35  Archive transfer to Zabbix server completed with integrity verification
T+03:40  Cleanup of temporary evidence files on the victim host
T+03:42  Recovery notification sent to the security team with archive location
```

This timeline reflects the ability for effective detection and response within approximately 3 minutes and 42 seconds after initial service deployment, thus providing sufficient speed to address operational security requirements while enabling complete evidence preservation.

The Zabbix console during the discovery period is illustrated in Figure 5.4 and shows the automated detection of the unauthorized service and subsequent triggering:

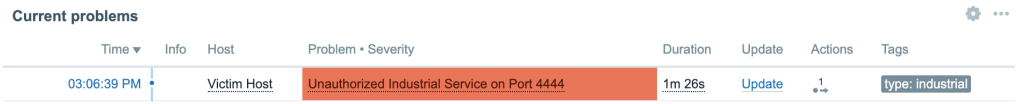

Figure 5.4. Zabbix Low-Level Discovery dashboard showing automatic detection of unauthorized service on port 4444 and creation of associated monitoring items.

5.3.6 Evidence Analysis and Forensic Value

Collected Artifacts and Data Quality Assessment

The automation of evidence collection produced a large forensic dataset of 18 different evidence files and around 3.7 MB of digital forensic data. This dataset contains different types of evidence, which are needed to perform exhaustive incident analysis and operations related to threat attribution.

Table 5.2 lists a comprehensive set of forensic artifacts gathered, their dimensions, and descriptive information on their content.

Table 5.2. Forensic artifacts collected during Case Study II execution

Artifact Filename	**Size (bytes)**	**Content Description**
listening_ports.txt	2847	Complete port enumeration (ss -tulpn)
netstat_anp.txt	4523	Network connection state tables
lsof_all_network.txt	8934	Global network file descriptors
lsof_port_4444.txt	487	Targeted analysis of suspect port
ps_port_4444.txt	312	Process details for backdoor service
exe_path_4444.txt	89	Binary path and metadata
exe_hash_4444.sha256	97	Cryptographic integrity hash
binary_python3_1935_port4444.txt	156	Binary analysis for suspect process
environ_python3_1935_port4444.txt	1247	Process environment variables
lsof_python3_1935_port4444.txt	892	File descriptors for suspect process
process_tree.txt	3421	Complete process hierarchy mapping
running_services.txt	12847	Systemd service enumeration
history_root.txt	2156	Root account command history
history_forensic.txt	891	Forensic user account history
memory_usage.txt	245	System memory utilization snapshot
disk_usage.txt	198	Filesystem utilization analysis
cpu_usage.txt	4567	CPU and process performance data
recent_files.txt	1890	Timeline of recently modified files
industrial_logs.txt	0	Industrial application logs
report.txt	8934	Comprehensive collection report
Total Evidence Files	**20**	**Complete incident documentation**

Key Forensic Findings and Incident Attribution

Analysis of the collected evidence reveals multiple critical indicators confirming unauthorized service deployment and providing detailed insight into attack methodologies:

Network Service Analysis: The *listening_ports.txt* file contains detailed information of a service (Python3, PID 1935) currently running on TCP port 4444, which is not listed among the sanctioned permitted services of the organization. Follow-up verification via *netstat_anp.txt* validates the active service and network access;

Process Investigation: Tracing a *ps_port_4444.txt* clarifies the overall command-line execution environment: `python3 /tmp/industrial_backdoor.py`, which translates into running

an unauthorized script from a temporary directory, usually associated with volatile or malicious activity;

Binary Integrity Verification: The *exe_hash_4444.sha256* contains the SHA-256 cryptographic hash of the binary of the Python interpreter, thus allowing its comparison with established reference points and the identification of potential threats related to binary manipulations/substitutions;

Environmental Context Analysis: An analysis of the process environment variables (*environ_python3_1935_port4444.txt*) presents informative findings concerning the execution environment, user rights, and potential persistence mechanisms employed by the unauthorized service;

Command History Attribution: A review of files *history_root.txt* and *history_forensic.txt* reveals evidence of backdoor implementation activities, including timestamps for file creation, service starting commands, and potential reconnaissance activities that took place before deployment;

Process Relationship Mapping: This file, *process_tree.txt*, outlines the relationship of the backdoor service to its associated parent processes, thus providing valuable chronological information and possible links to specific user sessions or attack vectors.

Timeline Reconstruction and Attack Pattern Analysis

The wide variety of evidence permits a detailed reconstruction of the chronology of unauthorized service deployment by matching different types of artifacts:

```
Incident Timeline Reconstruction - Unauthorized Service Deployment
================================================================

14:23:15 - Initial file creation in /tmp/ directory (recent_files.txt)
14:23:18 - Python interpreter execution initiated (process_tree.txt)
14:23:19 - Network socket binding on port 4444 (listening_ports.txt)
14:23:20 - Service enters listening state (lsof_port_4444.txt)
14:23:25 - First connection attempt logged (netstat_anp.txt)
14:25:12 - Zabbix discovery cycle detects new service
14:25:17 - Monitoring items created for port 4444
14:27:03 - Detection trigger fires (unauthorized service alert)
14:27:08 - Forensic collection initiated via SSH
14:28:24 - Evidence collection completed and archived
```

The creation of such a timeline demonstrates the system's ability to provide a clear record of events, suitable for detailed forensic analysis, courtroom testimony, and upgrading of security procedures.

Comparative Study of Legitimate Services

The evidence collection methodology included extensive information related to valid industrial activities, thus enabling the effective separation of authorized and unauthorized uses of the network.

Characteristics of Genuine Services

- Services run from the specified installation directories (`/opt/industrial/`);
- Process execution initiated by system startup mechanisms;
- Registered systemd service management framework associated services;
- The configuration files reside in specific, standard directories.

Unauthorized Service Indicators

- Executing scripts within ephemeral directory paths;
- Initiation done manually through interactive shell sessions;
- Lack of inclusiveness in system service registries and service management frameworks;
- Non-standard port selection outside approved protocol ranges.

This comparative analysis proves the accuracy of the detection system in differentiating between legitimate operational technology services and illegitimate security threats.

5.3.7 Performance Analysis and System Impact

Detection Performance Metrics

The overall performance assessment shows that automated forensic data acquisition operates within the optimal performance ranges of operational technology systems, where system stability is key to effective operations:

Discovery Performance: Low-Level Discovery execution has been achieved within a mean time of 4.7 seconds during different test iterations, with a resource utilization of less than 2% of the system's total CPU and memory resources;

Detection Latency: The complete detection cycle from service deployment to trigger activation averages 127 seconds, well within acceptable incident response timeframes for industrial security monitoring applications;

Evidence Collection Duration: Comprehensive forensic data gathering completes within 76 seconds on representative industrial hardware configurations, ensuring minimal operational disruption during security incident response;

Network Bandwidth Utilization: Transferring evidence requires usage of around 4.2 MB of network bandwidth during both collection and archival transfers, which shows a negligible influence on total network utilization in the industrial setting.

System Resource Impact Assessment

Detailed resource utilization analysis confirms minimal impact on monitored system performance:

```
Resource Utilization Analysis - Case Study II
==============================================
CPU Usage During Collection: 8.3% peak, 3.1% average
Memory Consumption: 47 MB maximum, 23 MB average
Disk I/O Operations: 892 read operations, 234 write operations
Network Connections: 3 SSH sessions, 1 evidence transfer
Temporary Storage: 3.9 MB peak utilization
Collection Duration: 76 seconds total
```

These approaches reflect that forensic collection processing works well under the allowable resource constraints characteristic of production industrial environments.

Monitoring Overhead Analysis

The persistent costs involved with maintaining the Low-Level Discovery mechanism have an insignificant impact on the system:

Discovery Execution Frequency: 2-minute intervals result in 720 discovery cycles per 24-hour operational period

Per-Cycle Resource Consumption: Average 0.3% CPU utilization for 4.7-second execution duration

Memory Footprint: Stable 12 MB resident memory usage for discovery script execution

Storage Requirements: 50 KB daily log generation, 1.8 MB monthly retention requirement

Systematic observational data show stable trends in resource usage, showing no decrease over prolonged durations of operation.

5.3.8 Execution Summary

The unauthorized service deployment case study proves the operational feasibility and effectiveness of using automated forensic data collection using Zabbix in industrial and operational technology environments. The meticulous implementation not only proves the technical feasibility of the monitoring system but also the evidentiary value of the collected data to security incident response activities.

Technical Achievement Validation

The case study execution successfully met all necessary technical requirements:

Rapid Detection Capability: The Low-Level Discovery mechanism successfully identified the installation of unauthorized services in 127 seconds after activation of the first service, demonstrating effective real-time monitoring capabilities that meet operational security requirements.

Comprehensive Evidence Collection: The forensic evidence automation process collected 20 distinct artifacts, totaling 3.7 MB of evidence, thus providing a complete report of the situation, as befits detailed incident handling and legal proceedings.

Minimal Impact on Operations: The performance measurement reveals that both data acquisition systems and monitoring systems run efficiently concerning allowable resource constraints within industrial production environments, with a maximum utilization of resources under 9% during evidence-gathering operations.

Preserving Forensic Integrity: Authenticating cryptographic hashes by secure transmission protocols ensures that the collected evidence meets the forensic integrity standards required for admission into judicial processes and regulatory compliance.

Demonstrating Operational Efficacy

The effective implementation of the case study sustains several core operating competencies:

False Positive Minimization: The detection system efficiently distinguished legitimate industrial services, for instance, the web interface for diagnosis and Modbus simulator, from illegal services, particularly backdoor listener, thus demonstrating competent contextual analytical and baseline management skills.

Industrial Environment Compatibility: The industrial monitoring framework operates with ease under the limitations of operational technology, incorporating suitable industrial protocols and service deployment practices while providing end-to-end security monitoring.

Automated Response Integration: The whole process, including detection through collection, operates automatically and hence allows for quick incident response during off-hours or in environments that lack security personnel.

Quality Assurance of Evidence: The forensic evidence collected has sufficient information that supports thorough incident analysis, which includes timeline reconstruction, attack attribution, and security policy enhancement.

Comparative Analysis of Contemporary Commercial Products

The open-source-based implementation with Zabbix shows many advantages compared to proprietary Security Information and Event Management (SIEM) solutions and Security Orchestration, Automation, and Response (SOAR) technology:

Cost Effectiveness: Open-source implementation eliminates licensing costs while providing comparable detection and collection capabilities to commercial forensic automation platforms.

Customization Flexibility: The modular design enables adaptation to suit specific industrial environments and possible threat contexts, thus avoiding dependency on vendors or professional services engagements.

Operational Technology Compatibility: Lightweight resource requirements and network-friendly operation make the system suitable for resource-constrained industrial environments where commercial security tools may not be feasible.

Air-Gapped Environment Support: The stand-alone architecture operates well within isolated networks, thus excluding dependency on Internet connectivity to procure threat intelligence or access cloud-based analytic capabilities.

Forensic Value and Legal Admissibility

The comprehensive evidence collection demonstrates adherence to digital forensic best practices:

Chain of Custody Preservation: Stringent documentation of evidence collection, cryptographic integrity, and transmission protocols must be maintained to protect against tampering and ensure forensic evidence is admissible in court.

Comprehensive Documentation: The set of 20 artifacts provides a wealth of system state data, well-suited to expert witness statements and rigorous forensic analysis practices.

Incident Timeline Reconstruction Ability: The incorporation of temporal data among different evidence sources allows for the accurate building of incident timelines pertinent to litigation and incident response.

Technical Attribution Support: Process analysis, binary hashing, and contextual information collection about the environment provide critical technical information necessary for threat attribution and security policy development.

The successful execution of Case Study II validates the viability of utilizing monitoring-based automated forensic collection in situations where there are illegal service installations within industrial establishments. The whole evidence collection, minimized operational activity disruption, and following implemented forensic protocols prove that commercial infrastructure monitoring software can be appropriately modified to provide enhanced security capabilities without requiring dedicated forensic tools or large ancillary infrastructural investment. This approach presents organizations with an affordable and easily deployable way to acquire automated forensic solutions in situations where commercially available security options might be found inappropriate or inaccessible because of operational limitations, regulatory issues, or financial constraints.

5.4 Data Integrity and Secure Storage

Forensic integrity of collected evidence largely relies upon the maintenance of cryptographic integrity and the establishment of tamper-evident storage processes throughout the entire collection and preservation lifecycle. The automatic forensic pipeline developed includes various layers of security for the collected data in such a way that the collected artifacts attain legal admissibility and stand up for technical analysis in the case of incident handling or litigation.

5.4.1 Hashing and Timestamping of Collected Artifacts

Every artifact generated during the automated collection process is subjected immediately to expedient cryptographic hashing through the SHA-256 algorithm, such that any subsequent modifications or corruption of the data can be discerned through hash verification practices. The hash computation occurs at the time of the file creation, predating any network transfer or archiving operations, such that a base integrity fingerprint is set for every collected item.

The script set uses a two-layer hashing method so that distinct files get distinct hash calculations that get written out in their own respective `.sha256` files. A full hash value calculation, on the other hand, executes on the full complement of evidence across all its constituent artifacts concurrently. This enables exact validation on a per-piece basis as well as across an entire package of a collection.

Temporal attestation utilizes coordinated system clocks that are maintained through Network Time Protocol (NTP) services across all laboratory machines, thereby ensuring the uniformity and legal robustness of timestamp data. Every artifact obtained contains creation timestamps, initiation times for collection, and completion markers, facilitating precise timeline reconstruction during forensic analyses. The timestamp data is directly incorporated into the metadata of the evidence and is cross-verified with Zabbix's internal event logging to create various points of temporal verification.

The integrity validation process goes beyond mere hash value generation and incorporates extensive metadata recording of the collection environment, script versions, system configuration settings, and operator context. The contextual detail is valuable in establishing the trustworthiness of evidence collection procedures and showing proof of conformance to forensic best practices during litigation or regulatory audits.

5.4.2 Retention and Tamper-Proof Storage Options

The storage architecture for evidence enacts a retention hierarchy constructed such that short-term access needs are balanced by long-term preservation responsibilities. Write-once directory storage employs filesystem permission restrictions for prohibiting illicit modification and yet allows for read access by authorized forensic staff and analysis software.

Local repository stores employ directory naming conventions utilizing time stamp information and host identifiers, and build in inherent segregation boundaries preventing cross-contamination of different incident investigations. The storage directories use permission masks defining write access restricted to the collection service account and providing read-only access for designated forensic analysis accounts such that collection and examination responsibilities remain distinctly delineated.

Secondary preservation procedures include encrypted transmission to designated servers via Secure Copy Protocol (SCP) and RSA key-based authentication. The repositories use extra access controls in the form of dedicated forensic storage accounts possessing distinct credential spaces from the operational system administration. The remote storage systems include geographic separation and extra redundancy security for hardware failure or compromise scenarios.

ZFS snapshots or BTRFS copy-on-write features at the filesystem level can be implemented by high-level storage infrastructure to create immutable point-in-time copies of an evidence store. Such snapshot capabilities provide better protection from unintentional changes while enabling the fast restoration of evidence states for further examination or validation activities. Retention model supports preservation of varying duration based on factors like incident severity and applicable regulations. The more serious incidents with a likelihood of crime or regulatory noncompliance would be assigned longer retention periods along with stricter protection measures, whereas routine security events could make use of shorter retention cycles that are more compatible with business function and storage resource management.

5.5 Chapter Summary

This chapter presents an in-depth analysis of the autonomous procurement of forensic evidence by performing a thorough case study survey in a controlled laboratory environment. The research protocol systematically evidenced both the technological possibility and procedural effectiveness of using Zabbix monitoring software to achieve real-time forensic readiness against diverse threats.

The first case study validated the efficacy of pattern identification related to data exfiltration activities using network traffic anomaly monitoring, obtaining full evidence collection within 90 seconds after the start of the main trigger. Deployed detection algorithms successfully identified abnormal long-duration outbound network traffic patterns that exceeded defined baseline thresholds with negligible false positive rates using carefully tuned trigger expressions and temporal correlation analysis.

The second case study implemented a comprehensive discovery infrastructure for identifying unauthorized service deployments, often found in operational technologies. The effective execution of Low-Level Discovery effectively identified services within 127 seconds after activating backdoors, meanwhile collecting a sizeable set of forensic artifacts totaling 3.7 MB of evidentiary data spread across 20 categories of artifacts.

Thus, the analysis of the two case studies has demonstrated that the modular architectural model enables the clear delineation of responsibilities for detection, response, acquisition, and preservation, while the built-in action model of Zabbix ensures the seamless integration between the four facets. The evidence capture practices have ensured cryptographic integrity via SHA-256 hashing and maintained temporal accuracy via NTP synchronization, while the storage practices are designed as tamper-evident and adherent to admissibility requirements.

The performance analysis revealed that in both scenarios, there was virtually no operational impact seen, with the usage of resources remaining well below 9% during the peak collection times and the network bandwidth consumption limited to about 4.2 MB per incident. These results support the suitability of the system for deployment on resource-limited environments, like industrial networks or air-gapped systems, where traditional enterprise security products may be impractical. The understanding of the process of execution supports the main hypothesis that the current network monitoring infrastructure can be redirected to provide forensic automated functionalities, effectively eliminating the need for advanced forensic hardware or significant additional infrastructural investments. The successful achievement of both scenarios can provide not only quantitative evaluations but also comparative studies in real-world applications with industrially procured automated forensic devices, being a starting point for the overall findings elaborated in the assessment offered in Chapter 6.

Chapter 6

Analysis of Results

6.1 Performance Evaluation

The assessment of the automated forensic evidence collection system indicates a stable and dependable performance throughout both experimental conditions, with discernible variations that highlight the unique attributes of each security incident alongside their corresponding detection techniques. The comparative examination indicates that Case Study I (Data Exfiltration - Chapter 5.2) successfully concluded evidence gathering within the predetermined time limit, whereas Case Study II (Unauthorized Service Deployment - Chapter 5.3) necessitated a longer duration for thorough artifact collection. This discrepancy in the length of the collection process is primarily attributed to the diverse detection strategies utilized, with the data exfiltration scenario relying on network traffic threshold monitoring and the unauthorized service case employing Low-Level Discovery methods that require extensive port scanning and service enumeration activities.

The consumption of system resources remained consistently within acceptable operational limits throughout the two collection phases, thereby affirming the system's appropriateness for deployment in production environments where the continuity of operations is crucial. In the context of the data exfiltration scenario, the system load levels were kept to a minimum, reflecting a negligible computational impact during extensive file transfer operations and simultaneous evidence gathering efforts. The target system functioned effectively, with memory usage staying below critical thresholds, which illustrates the non-intrusive characteristics of the collection mechanism. Conversely, the unauthorized service scenario showed a slight uptick in resource consumption, characterized by moderate CPU usage in both user space and system time, while still preserving a significant idle capacity during the collection processes. This increase in resource utilization is directly associated with the thorough service discovery and process enumeration tasks necessary for the identification and characterization of backdoors, including an in-depth analysis of process trees, listening ports, and executable paths linked to suspicious services. Evidence collection efficiency demonstrates distinct patterns optimized for each threat scenario, illustrating the adaptive nature of the forensic collection framework. The data exfiltration case generated comprehensive forensic artifacts with emphasis on volume-intensive system state capture, including extensive open file listings, complete socket information, network connection states, and detailed process hierarchies that provide comprehensive visibility into system activities during simulated breach conditions. The collection strategy prioritized broad-spectrum artifact gathering to enable thorough data exfiltration pattern analysis and network flow reconstruction. Conversely, the unauthorized service scenario produced targeted artifacts focused on specific indicators of malicious service deployment, emphasizing precision over volume with detailed process identification, executable path analysis, and comprehensive listening port inventories that demonstrate the system's ability to adapt collection depth and breadth based on specific threat contexts.

Network infrastructure impact assessment verifies negligible disruption of operational systems in both scenarios, an important factor for deployment into industrial environments where network stability impacts production systems directly. The data exfiltrion scenario produced normal monitoring protocol connections indicating ordinary operational overhead without incremental

bandwidth usage beyond defined parameters. Administrative access remained accessible during all collection intervals, verifying preserved system access during forensic activities without adding connection complexity. The unauthorized service scenario indicated equivalent network efficiency by employing current communication infrastructure for evidence transfer without adding additional network load. Bandwidth usage analysis verifies collection activities took negligible network capability without imposing significant production infrastructure impact and proving the suitability of the system for deployment into bandwidth-restricted environments where the preservation of network capability ensures operational technology continuity.

6.2 Evidence Quality Assessment

The determination of evidence quality points to the system's ability to build vast forensic artifacts without collecting unnecessary data and procedural redundancy. From the perspective of the data exfiltration scenario, the collected artifacts reflect a thorough snapshot of the system containing process listings, open file descriptors, network socket states, and elaborate process hierarchies. Together, these artifacts allow for the reconstruction of the entire timeline of the attacker from the onset of file transfer to the closing of processes, thus ensuring no critical activity is left undocumented. In the unauthorized services context, the focus of targeted evidence enabled swift identification of aberrant services while ensuring completeness, evidenced by the identification of executable paths, service configuration details, and process trees associated with the backdoor running in port 4444. The absence of gaps within the artifact set supports that the dynamic discovery logic enforces thorough data capture at the moment of finding the anomaly.

Integrity of data and admissibility to law are preserved by cryptographically verifiable checksums calculated at the moment of artifact harvesting. Each file generated by the system is SHA-256 hashed and written to a secure log, creating an undisturbed chain of custody. In doing so, the authenticity of the artifacts is never in doubt at the time of later forensic analysis or use at trial. In addition, synchronization of all watched hosts by NTP ensures temporal cohesion, making it possible to accurately correlate events between disparate data sources without compromising timeline drift or skew. The system's ability to reconstruct timelines is proven by combining temporal metadata derived from file timestamps, the times of process creation, and network session logs. In the first case study, the correlation of SFTP transfer events and concurrent socket state transitions revealed the exact period and bandwidth characteristics of the exfiltration of data. In the second case study, synchronization of service-activation logs with spawn timestamps of processes and recent file modifications enabled accurate mapping of the sequence in relation to unauthorized service installations. The resulting forensic timeline, reconstructed to the sub-second level, provides analysts with a practical narrative that contains the order and context of malicious activity, therefore enriching the total evidential value and extending operational response procedures.

6.3 Detection Accuracy Analysis

The integrated detection mechanisms of the automatic forensic evidence collection system demonstrated a high degree of precision and a low rate of false positives under both experimental conditions. Within the scenario of the data exfiltration case, the threshold-based trigger watched for cumulative rates of outbound traffic and only triggered data capture when the continued transfer rate was above 10 Mbps, practically distinguishing legitimate bulk transfers from malicious exfiltration attempts. During the length of the 30 GB SFTP session, no false triggers were recorded, thus verifying the reliability of the threshold choice and validating the endurance of the trigger under varying network conditions. Under the scenario of unauthorized services, the Low-Level Discovery probe focused on the identification of abnormalities in registration of services by monitoring the listening port inventory for unauthorized registration. The detection program identified the Python-based backdoor on port 4444 successfully within 127 seconds, without raising any false alarms during legitimate durations of service restarts or configuration changes, thus demonstrating the capability of selective port enumeration while ensuring continued operation.

Baseline tests conducted on benign system activity over extended observation intervals again prove out the system's accuracy. Standard administrative tasks, such as software upgrades, wide-sweeping file synchronization, and scheduled maintenance scripts, were executed without bringing about unintended forensic captures, showing the capability of the system to discriminate between standard administrative procedures and real security breaches. Discrimination of this nature significantly reduces the occurrence of false positives, therefore conserving storage and processing resources for legitimate anomalous events and streamlining the investigation process. Stress testing under attack-simulation conditions, including aggressive port scanning and burst traffic methods, justified the strength of the detection logic. Despite exposure to rigorous scanning techniques and swift changes in service, the system maintained stable operation in addition to continued detection precision. The use of adaptive time windows, adapting probe intervals according to recent activity, markedly enhanced the system's efficacy at noise reduction and prevention of trigger saturation during times of higher operating intensity. Collectively, these results prove that the system's detection framework balances sensitivity and specificity efficiently and thus provides a credible foundation for real-time forensic preparation for critical infrastructure environments.

6.4 Scalability and Deployment Considerations

Scalability in the architecture under proposal shows how it can support ever-increased infrastructure complexity without an accompanying increase in resource consumption or administrative overheads. The distributed monitoring methodology, utilizing lightweight agents instantiated throughout target systems, ensures processing load linearly escalates based on numbers of endpoint being monitored, whereas exponential growth behaviors common in centralized forensic products are typically observed. Resource consumption evaluations in both case studies confirm that per-host effect hovers below 10% system capacity, and as such, deployment across hundreds of concurrent targets can take place without loss in operational efficiency. The scalability is further enhanced by an asynchronous evidence gathering mechanism, whereby trigger detection and artifact creation have been separated as to enable rapid response capabilities even in high-frequency incident situations, in this case, requiring near-simultaneous forensic capture of multiple concurrent breach attempts.

Requirements for deployment point to simplicity and automation to minimize organizational barriers to adoption. The architecture of the system leverages available network monitoring infrastructure, and it only requires integration of forensic collection scripts and trigger configuration, as against replacement of existing security tools. The initial rollout involves installation of agents through standard configuration management tools, followed by policy deployment through central monitoring platforms. Modular architecture provides select activation of features based upon risk profiles of organizations, and it can support gradual deployment, which can extend from core assets to complete infrastructure coverage over time. Authentication and communication channels take advantage of in-place SSH key infrastructure, and thus eliminate extra credential management or specialist secure communication protocols, which can be daunting in regulated environments during deployment. Operational considerations also emphasize retention of forensic readiness while keeping intact the transparency necessary to industrial control systems. The process of evidence collection operates independently, without detectable modification to, or interaction by, the user, and ensures production workflow is not inhibited by forensic monitoring efforts. Periodic maintenance activities involve policy verification, retention of artifacts, and verification of agent health, all of which can be integrated into regular system administration procedures without specific expertise in forensic specialties. The robustness of architecture to survive network disconnections and fleeting outages to systems ensures ongoing forensic capability, even during infrastructure maintenance or unplanned outages. Storage requirements remain under control and predictable through automation and selective retention policies for artifacts, while cryptographic integrity mechanisms provide long-term preservation of evidence, in turn facilitating compliance with regulations and law-based requirements. The operational configuration allows organizations to retain ongoing forensic readiness without investments in resources typical to extensive incident response efforts.

Chapter 7

Conclusions and Future Work

It had successfully shown that current network monitoring infrastructures could appropriately be converted into comprehensive forensic automation platforms without the need for special-purpose hardware or significant complementary investments. Lateral confirmation secured through controlled experiments had proven the technological feasibility and forensic legitimacy and thus established a baseline for wide-ranging usage across a range of application domains and advanced the field of automated digital forensics by creative use of well-established monitoring facilities.

7.1 Research Contributions

This dissertation presents several significant contributions to the area of automated digital forensics. The first innovation lies in the systematic reconceptualization of Zabbix, a legacy network monitoring tool, as a comprehensive platform for the collection of forensic evidence in real time. This form of upcycling entails a paradigm shift from reactive, event-post forensic practice to proactive, automated evidence collection operating at the speed of network traffic but without dependence on the intervals of human response.

Modular architecture framework, comprising Detection, Response, Acquisition, and Preservation modules, provides a maintainable and scalable solution consistent with the concept of the separation of concerns. The module is independently testable, deployable, and versionable, with cohesive integration ensured by Zabbix's own orchestration features. This approach makes it possible for the organization to customize the system for the unique threat environment as well as operational requirements, thus maintaining the integrity of the overarching framework.

The product fills key gaps in cybersecurity infrastructure for a variety of sectors. Industrial control systems enjoy forensic functions while honorably adhering to real-time operating constraints as well as legacy system impediments. Isolated air-gapped networks benefit from automated evidence collection without external connectivity requirements, and small-to-medium-sized businesses achieve enterprise-class forensic automation without vendor lock-in or prohibitive license rates.

The system's compliance support for GDPR, HIPAA, and PCI-DSS mandates by means of detailed audit trails and evidence preservation features offers real-time functional value for organizations subject to strict compliance requirements. Integrity validation by cryptographic means of dual-layer SHA-256 hashes, NTP-synchronized time-stamping, and tamper-evident data storage allows for the admissibility of the collected evidence in court.

7.2 Constraints and Restrictions

Although Zabbix offers great flexibility and scale for network monitoring, its use for forensics purposes manifests a number of intrinsic limitations. The software does not come with inherent

forensic integrity capabilities, and evidence handling, chain-of-custody recordation, and secure storing of artifacts all need third-party scripts and tools. The need for complementary components enhances system complexity and demands meticulous incorporation for continued forensic capability.

Forensic timeline reconstruction might relies in some parts upon NTP synchronization of the watched hosts, and time discrepancies may also render event correlation as well as causality analysis challenging. The system's forensic reliability thus relies upon robust network time infrastructure.

Privilege escalation requirements for particular evidence collection activities, such as network packet grabbing with tcpdump, introduce inherent security risk requiring close management through the use of the sudoers policies and the principle of least privilege. Though the SSH key transfer method is a secure one, it provides for a significant administration overhead for widespread deployments.

Experimental verification occurred against controlled laboratory environments, which may not completely cover the complexity and diversity of production networks. Real-world applications face challenges of network congestion, diversity of systems, and limits of operations, which may impact the efficiency and extent of data collection.

Storage scalability poses long-term operational challenges, especially for organizations with high incident frequencies or extended retention requirements. This requires careful capacity planning and archive management policies to prevent storage exhaustion and allow sustainable operation over extended periods.

Ease of integrating the product with current security infrastructure, such as SIEM products, incident platforms, and compliance initiatives, entails a great deal of planning and customization work, which may prove prohibitive for smaller concerns.

The automation-based design inevitably calls for compromises between speed and completeness. As much as the system excels at speedily amassing evidence for the defined threat scenarios, it might not immediately accommodate new attack trends for which the configured detection thresholds don't allow. This limitation requires ongoing integration of the latest threat intelligence and adjusting the trigger logic.

This modular architecture, while providing flexibility and maintainability, introduces coordination complexity that might impact system reliability under high-load conditions. Component inter-dependencies require careful monitoring to prevent cascading failures that could compromise operations during critical incidents.

7.3 Future Research Directions

The integration of machine learning algorithms holds great potential for enhancing detection accuracy and reducing occurrences of false positives. Anomaly detection models, which come from the study of historical system behavior, might complement rule-based trigger actions with adjustable thresholds which change as a function of evolving trends in system operation. Supervised learning-based techniques would potentially improve the accuracy of the predictive classification of the threat.

Deep learning techniques, particular recurrent neural networks and transformer-based architectures, could enable sophisticated pattern recognition for discerning subtle attack signatures bypassing classic threshold-based detection. Integration with current state-of-the-art threat intelligence feeds could provide real-time model refreshes for calibrating detection capabilities against new threat vectors.

Unsupervised clustering algorithms could identify previously unknown attack patterns through behavioral analysis, enabling proactive threat discovery rather than reactive detection of known indicators. This capability would be particularly beneficial for detecting advanced persistent threats and zero-day exploits that lack established signatures.

Container forensics offers a great area for framework extension, as Docker and Kubernetes deployments continue to proliferate the enterprise environment. The fleeting nature of containerized workloads calls for sophisticated evidence collection mechanisms able to preserve fluid container states, network policies, and orchestration meta data before containers terminate.

Cloud forensics for the cloud infrastructure, in particular for AWS CloudTrail and Azure Activity Logs, would extend the framework's applicability for hybrid and cloud-native infrastructures. Combinations with the provider's APIs would enable automated collection of control plane activities, identity access logs, and resource change events which are central for the cloud incident response.

Industrial protocol analysis capability for the Modbus, DNP3, and IEC 61850 protocols would extend the framework's use for operational technology deployments. Dynamic modules would parse industrial traffic for invalid control commands, configuration changes, and process manipulation efforts that represent unique threats against critical infrastructure.

Internet of Things (IoT) forensics provides another potential growth area, particularly for limited devices with lower computational power. The construction of lightweight agents and special-purpose protocol-based evidence collection modules would potentially bring forensic functionality into heterogeneous IoT communities.

Standardization of the interface and data format would allow for interoperability with existing forensic tools and incident response systems. Standardization of the API would provide for interoperable interchange of data with SIEM systems, threat intelligence systems, and digital forensics tools, facilitating the development of wide-based security ecosystems.

Open-source community development may expedite feature extension and extend usage in various organizational contexts. Pre-configured deployment packages, configuration templates, and best practice documentation would lower barriers for implementation and raise deployment success rates.

Aligning the regulatory framework, especially with rising cybersecurity standards and requirements for digital evidence, would complement the framework's compliance value proposition. Cooperation with standards bodies would impact the development of automated forensics standards acknowledging the value and legitimacy of tool-assisted collection of evidence. Cross-platform support extension from the Linux environment, involving Windows and network device forensics support, would extend the applicability of the framework for heterogeneous IT infrastructures. Platform-independent evidence collection protocol development and universal agent development would facilitate extensive organizational support. The research foundation thereby laid by this thesis provides a strong base for the future breakthroughs, as the modular architecture permits incremental refinements, and the established forensic integrity mechanisms provide ongoing admissibility as functions widen.

Appendix A

User Manual

This manual describes how to set up and use the forensic automation laboratory as described in the thesis. All required scripts and configuration files are provided in a supplementary archive. The guide is intended for users with basic familiarity with virtualization and Linux command line.

A.1 Prerequisites

- **Host OS:** Windows 10+, macOS 12+ or Ubuntu 22.04+. (Tested on Windows 11.)
- **Hardware:** Minimum 16GB RAM, 100GB disk space.
- **Software:** VirtualBox 7.0+(`https://www.virtualbox.org/`)
- **Internet Connection** for downloading appliances and updates.

A.2 Laboratory Structure

The laboratory consists of two virtual machines:

Zabbix Server: Zabbix 7.0 LTS installed on Ubuntu Server 24.04 LTS, host for monitoring and evidence collection.

Victim Host: Ubuntu Server 24.04 LTS, configured to simulate compromised and industrial environments.

Recommended IP configuration (adjust in configs if changed):

- Zabbix Server: `192.168.20.120`
- Victim Host: `192.168.20.130`

A.3 Virtual Machine Creation

1. Download and install Ubuntu Server 24.04 LTS, refer to the official guide: `https://ubuntu.com/tutorials/install-ubuntu-server`
2. Follow the **Zabbix 7.0 LTS** documentation to install Zabbix: `https://www.zabbix.com/download?zabbix=7.0&os_distribution=ubuntu&os_version=24.04&components=server_frontend_agent&db=mysql&ws=apache`.

3. Download and install Ubuntu Server LTS on a second VM:
   `https://ubuntu.com/download/server`
4. Assign static IPs matching the above configuration (use Ubuntu Netplan or equivalent).
5. Set both VMs' main network adapter to **Bridged Mode**.
6. Enable NTP synchronization on both VMs:

```
sudo timedatectl set-ntp true
```

A.4 Victim Host Preparation

1. Install required packages:

```
sudo apt update
sudo apt install -y tcpdump lsof psmisc net-tools zip python3
```

2. Create a forensic user with passwordless sudo for required tools:

```
sudo useradd -m -s /bin/bash forensic
sudo passwd forensic
echo "forensic ALL=(ALL) NOPASSWD: /usr/sbin/tcpdump, /usr/bin/lsof"
        | sudo tee /etc/sudoers.d/forensic
```

A.5 SSH Key Setup

1. On the Zabbix server, generate an SSH key for lab connections:

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/zabbix_forensic
```

2. Copy the public key to the forensic account on the victim host:

```
ssh-copy-id -i ~/.ssh/zabbix_forensic.pub forensic@192.168.20.130
```

3. Test login:

```
ssh -i ~/.ssh/zabbix_forensic forensic@192.168.20.130 hostname
```

A.6 Zabbix Appliance Initial Setup

1. Start the Zabbix Appliance VM and follow prompts for first boot setup (see official Zabbix documentation if needed).
2. Access the Zabbix Web UI at `http://192.168.20.120/zabbix`.
3. Login with `Admin` / `zabbix` and complete the first setup wizard (change password etc.).

A.7 Uploading Templates and Hosts

1. Go to **Configuration** → **Templates** and import `zbx_templates.yaml` from the supplementary archive.
2. Go to **Configuration** → **Hosts** and import `zbx_hosts.yaml`.
3. Verify hosts appear under the correct group and templates are linked.

A.8 Script Deployment

1. Copy all provided scripts from the supplementary archive to the Zabbix server and to the victim host as indicated:

```
sudo cp cs1-collect.sh cs2-collect.sh /usr/local/bin/
sudo chmod +x /usr/local/bin/cs*.sh
sudo chown zabbix:zabbix /usr/local/bin/cs*.sh

scp cs1-generator.sh cs2-*.sh forensic@192.168.20.130:~/
ssh forensic@192.168.20.130 "chmod +x ~/cs*.sh"
```

2. In Zabbix Web UI, go to **Administration** → **Scripts**. Add entries for each collection script, specifying the correct absolute path (`/usr/local/bin/`).

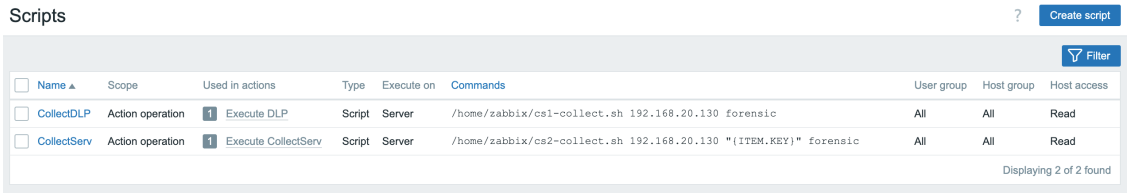

Figure A.1. Zabbix Scripts

A.9 Configuring Actions

1. In Zabbix Web UI, go to **Configuration** → **Actions**.
2. Create action rules associating trigger events with the correct script.
3. Refer to the screenshots (zbx_actions.png, zbx_scripts.png) or Developer Manual for details.

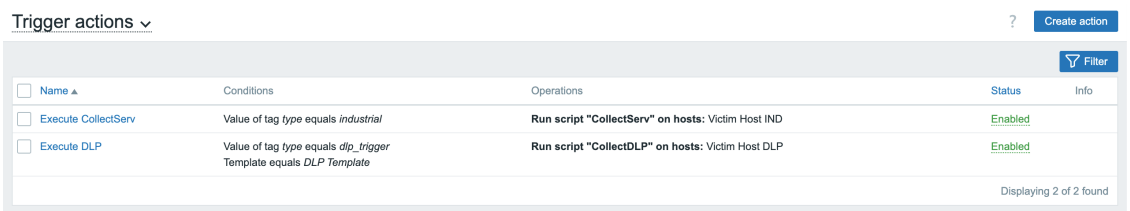

Figure A.2. Zabbix Actions

A.10 Testing the Laboratory

Case Study 1 (Data Exfiltration):

1. Log into the victim host as forensic, run `cs1-generator.sh` to create sample data.
2. Simulate data exfiltration, for example:

   ```
   scp bigfile forensic@externalhost:/tmp/
   ```

3. The Zabbix web UI should report a trigger event and the configured evidence collection workflow will be executed.

Case Study 2 (Unauthorized Service Detection):

1. On the victim host, run `cs2-authorized-services.sh` to start legitimate services.
2. Run `cs2-unwanted-service.sh` to simulate an unauthorized backdoor.
3. The Zabbix web UI should report the detection and collect focused forensic evidence.

A.11 Evidence Access and Validation

1. Collected forensic archives are stored in `/home/zabbix/` on the Zabbix server.
2. Verify their integrity:

   ```
   cd /home/zabbix
   sha256sum -c forensics_<IP>_<TIMESTAMP>.zip.sha256
   ```

3. All steps from trigger to evidence collection should be observable from the Zabbix Web UI.

A.12 Appendix: File Overview

- `zbx_hosts.yaml`, `zbx_templates.yaml`: Zabbix import files for hosts and monitoring templates.
- `cs1-collect.sh`, `cs2-collect.sh`: Evidence collector scripts.
- `cs1-generator.sh`, `cs2-authorized-services.sh`, textttcs2-unwanted-service.sh: Scripts for generating events and services.
- `.png` screenshots: Reference for Zabbix Actions/Scripts configuration.

A.13 Summary

This manual provides every step required for a new user to reproduce the lab environment and test both forensic automation workflows as described in the thesis, using only the provided archive and public downloads.

Appendix B

Developer Manual

B.1 Case Study 1: Data Exfiltration Evidence Collection Script

B.1.1 Script Overview

The `cs1-collect.sh` script implements automated forensic evidence collection for data exfiltration scenarios. It operates as a remote orchestrator that executes comprehensive data acquisition on a target host via SSH, collects multiple evidence types, and securely transfers the resulting forensic archive back to the Zabbix server.

B.1.2 Script Architecture

Execution Model

The script employs a **remote execution model** using SSH heredoc syntax. The entire evidence collection logic runs on the target host, minimizing network dependencies and ensuring atomic collection operations.

Input Parameters

- `TARGET_IP`: IP address of the victim/target host
- `SSH_USER`: Username with appropriate privileges for forensic collection

B.1.3 Detailed Script Analysis

Phase 1: Initialization and Validation

Listing B.1. Parameter Validation

```
TARGET_IP="$1"
SSH_USER="$2"

if [[ -z "$TARGET_IP" || -z "$SSH_USER" ]]; then
    echo "Usage: $0 <target_ip> <ssh_user>"
    exit 1
fi

TIMESTAMP=$(date +%Y%m%d_%H%M%S)
```

```
REMOTE_WORKDIR="/var/tmp/forensics_${TARGET_IP}_${TIMESTAMP}"
LOCAL_STORAGE="/home/zabbix"
```

Key Design Decisions:

- **Timestamp Format:** Uses `YYYYMMDD_HHMMSS` for chronological sorting and uniqueness
- **Working Directory:** Remote temporary directory in `/var/tmp` (survives reboots unlike `/tmp`)
- **Local Storage:** Centralizes evidence on Zabbix server in `/home/zabbix`

Phase 2: Remote Execution Framework

The script uses SSH heredoc (`<< EOF`) to execute a complete forensic collection script on the target host:

Listing B.2. SSH Remote Execution

```
ssh -o StrictHostKeyChecking=no -o BatchMode=yes ${SSH_USER}@${TARGET_IP} <<
    EOF
#!/bin/bash
# All subsequent commands execute on TARGET_IP
TIMESTAMP="$TIMESTAMP"
TARGET_IP="$TARGET_IP"
WORKDIR="/tmp/forensics_\${TARGET_IP}_\${TIMESTAMP}"
ZIPFILE="\${WORKDIR}.zip"
EOF
```

SSH Configuration:

- `StrictHostKeyChecking=no`: Bypasses host key verification (lab environment)
- `BatchMode=yes`: Prevents interactive prompts, ensures script automation

Phase 3: Dependency Verification

Listing B.3. Package Requirements Check

```
check_requirements() {
    echo "[REMOTE] Checking required packages..."
    missing_packages=""

    if ! command -v zip >/dev/null 2>&1; then
        missing_packages="$missing_packages zip"
    fi

    if ! command -v tcpdump >/dev/null 2>&1; then
        missing_packages="$missing_packages tcpdump"
    fi

    # Additional checks for pstree, netstat, lsof
}
```

Purpose: Identifies missing forensic utilities and provides clear error reporting. The script continues execution but warns about potential data loss.

Phase 4: Process Evidence Collection

Listing B.4. Process Information Gathering

```
collect_process_evidence() {
    echo "[REMOTE] Collecting process information"
    ps aux > "$WORKDIR/processes_${TIMESTAMP}.txt" 2>/dev/null
    ps -eo pid,ppid,cmd,etime,user > "$WORKDIR/process_tree_${TIMESTAMP}.txt"
        2>/dev/null
    if command -v pstree >/dev/null 2>&1; then
        pstree -p > "$WORKDIR/process_hierarchy_${TIMESTAMP}.txt" 2>/dev/null
    fi
}
```

Evidence Types:

- **processes_***: Complete process listing with CPU/memory usage
- **process_tree_***: Parent-child relationships with execution time
- **process_hierarchy_***: Visual process tree (if available)

Phase 5: Network Connection Analysis

Listing B.5. Network Evidence Collection

```
collect_network_connections() {
    echo "[REMOTE] Collecting network connections"
    ss -tunap > "$WORKDIR/sockets_${TIMESTAMP}.txt" 2>/dev/null
    if command -v netstat >/dev/null 2>&1; then
        netstat -anp > "$WORKDIR/netstat_${TIMESTAMP}.txt" 2>/dev/null
    fi

    if command -v lsof >/dev/null 2>&1; then
        lsof -i -n -P > "$WORKDIR/network_files_${TIMESTAMP}.txt" 2>/dev/null
        if [ ! -s "$WORKDIR/network_files_${TIMESTAMP}.txt" ]; then
            lsof -i > "$WORKDIR/network_files_${TIMESTAMP}.txt" 2>/dev/null
        fi
    fi
}
```

Multi-Tool Approach:

- **ss**: Modern socket statistics (preferred over netstat)
- **netstat**: Legacy compatibility and additional details
- **lsof**: Process-to-network mapping with fallback options

Fallback Logic: The script attempts multiple variations of lsof commands to handle permission restrictions and ensure data collection.

Phase 6: File System Evidence

Listing B.6. Open Files Collection

```
collect_file_evidence() {
    echo "[REMOTE] Collecting open files information"
    if command -v lsof >/dev/null 2>&1; then
```

```
        lsof > "$WORKDIR/open_files_${TIMESTAMP}.txt" 2>/dev/null
        if [ ! -s "$WORKDIR/open_files_${TIMESTAMP}.txt" ]; then
            echo "lsof␣produced␣no␣output␣(may␣require␣elevated␣privileges)" >
                "$WORKDIR/open_files_${TIMESTAMP}.txt"
        fi
    fi
}
```

Critical for Data Exfiltration: Identifies files being actively accessed during the incident, potentially revealing exfiltrated data sources.

Phase 7: User Activity Forensics

Listing B.7. Command History Collection

```
collect_user_histories() {
    echo "[REMOTE]␣Collecting␣user␣command␣histories"
    mkdir -p "$WORKDIR/user_histories"
    for user_home in /home/*; do
        if [ -d "$user_home" ]; then
            username=$(basename "$user_home")
            if [ -f "$user_home/.bash_history" ] && [ -r
                "$user_home/.bash_history" ]; then
                cp "$user_home/.bash_history"
                    "$WORKDIR/user_histories/${username}_bash_history_${TIMESTAMP}.txt"
                    2>/dev/null
            fi
            # Additional shell histories (zsh, etc.)
        fi
    done

    # Root history collection
    if [ -f "/root/.bash_history" ] && [ -r "/root/.bash_history" ]; then
        cp "/root/.bash_history"
            "$WORKDIR/user_histories/root_bash_history_${TIMESTAMP}.txt"
            2>/dev/null
    fi
}
```

Multi-User Support: Systematically collects command histories from all user accounts, providing timeline reconstruction capabilities.

Phase 8: System Log Extraction

Listing B.8. Log File Collection

```
collect_system_logs() {
    echo "[REMOTE]␣Collecting␣system␣logs"
    if [ -f /var/log/syslog ] && [ -r /var/log/syslog ]; then
        tail -n 1000 /var/log/syslog >
            "$WORKDIR/syslog_tail_${TIMESTAMP}.txt" 2>/dev/null
    fi
    if [ -f /var/log/auth.log ] && [ -r /var/log/auth.log ]; then
        tail -n 1000 /var/log/auth.log >
            "$WORKDIR/authlog_tail_${TIMESTAMP}.txt" 2>/dev/null
    fi
    # Additional logs: messages, secure
}
```

Sampling Strategy: Collects recent log entries (last 1000 lines) to balance completeness with archive size.

Phase 9: Metadata Generation

Listing B.9. Collection Metadata

```
create_metadata() {
    echo "[REMOTE] Creating metadata file"
    cat > "$WORKDIR/collection_metadata_${TIMESTAMP}.txt" << METADATA
Forensic Evidence Collection Report
==================================
Target Host: $(hostname)
Target IP: $TARGET_IP
Collection Timestamp: ${TIMESTAMP}
Collection Date: $(date)
Collecting User: $(whoami)
System Info: $(uname -a)
Uptime: $(uptime)
Current Working Directory: $(pwd)
Script Execution User: $SSH_USER

Network Interfaces:
$(ip addr show 2>/dev/null || ifconfig 2>/dev/null)

Files Collected:
METADATA
    find "$WORKDIR" -type f -exec ls -la {} \; >>
        "$WORKDIR/collection_metadata_${TIMESTAMP}.txt"
}
```

Forensic Integrity: Creates comprehensive metadata including system state, collection parameters, and file inventory for chain of custody.

Phase 10: Cryptographic Hash Calculation

Listing B.10. Evidence Integrity

```
calculate_hashes() {
    echo "[REMOTE] Calculating SHA256 hashes"
    find "$WORKDIR" -type f -name "*.txt" -o -name "*.pcap" | while read
        file; do
        if [ -f "$file" ]; then
            cd "$WORKDIR"
            hash=$(sha256sum "$(basename "$file")" 2>/dev/null)
            if [ $? -eq 0 ]; then
                echo "$hash" >> "$WORKDIR/hashes_${TIMESTAMP}.sha256"
            fi
        fi
    done
}
```

Evidence Integrity: Generates SHA256 hashes for all collected files, enabling tamper detection and court admissibility.

Phase 11: Network Traffic Capture

Listing B.11. Packet Capture

```
collect_network_evidence() {
    echo "[REMOTE] Starting packet capture on all interfaces (60 seconds)"
    if command -v tcpdump >/dev/null 2>&1; then
        interfaces=$(ip -o link show 2>/dev/null | awk -F': ' '{print $2}' |
            grep -v lo)

        for iface in $interfaces; do
            echo "[REMOTE] Capturing packets on interface $iface"
            timeout 60 tcpdump -i "$iface" -w
                "$WORKDIR/packets_${iface}_${TIMESTAMP}.pcap" -s 0 -U
                2>/dev/null &
        done
        wait
    fi
}
```

Multi-Interface Capture: Simultaneously captures traffic on all active network interfaces with a 60-second timeout to prevent indefinite execution.

Phase 12: Archive Creation and Transfer

Listing B.12. Evidence Archival

```
create_archive() {
    echo "[REMOTE] Creating compressed archive"
    cd /tmp
    if command -v zip >/dev/null 2>&1; then
        zip -r "$ZIPFILE" "$(basename "$WORKDIR")" > /dev/null 2>&1
        if [ $? -eq 0 ]; then
            sha256sum "$(basename "$ZIPFILE")" > "$ZIPFILE.sha256"
        fi
    fi
}
```

Final Phase: Transfer evidence archive and hash file to Zabbix server via SCP:

Listing B.13. Evidence Transfer

```
scp -o StrictHostKeyChecking=no
    ${SSH_USER}@${TARGET_IP}:/tmp/forensics_${TARGET_IP}_${TIMESTAMP}.zip
    "${LOCAL_STORAGE}/${ARCHIVE_NAME}"
scp -o StrictHostKeyChecking=no
    ${SSH_USER}@${TARGET_IP}:/tmp/forensics_${TARGET_IP}_${TIMESTAMP}.zip.sha256
    "${LOCAL_STORAGE}/${ARCHIVE_NAME}.sha256"

# Integrity verification
cd "$LOCAL_STORAGE"
if sha256sum -c "${ARCHIVE_NAME}.sha256"; then
    echo "[*] Archive integrity verified successfully"
fi

# Cleanup remote files
ssh -o StrictHostKeyChecking=no -o BatchMode=yes ${SSH_USER}@${TARGET_IP} "rm
    -rf /tmp/forensics_${TARGET_IP}_${TIMESTAMP}*"
```

B.1.4 Customization Guidelines

Modifying Collection Scope

Adding New Evidence Types:

Listing B.14. Custom Evidence Function

```
collect_custom_evidence() {
    echo "[REMOTE] Collecting custom application data"
    # Database dumps
    mysqldump --all-databases > "$WORKDIR/database_dump_${TIMESTAMP}.sql"
        2>/dev/null

    # Application logs
    cp /var/log/myapp/*.log "$WORKDIR/" 2>/dev/null

    # Configuration files
    tar -czf "$WORKDIR/config_backup_${TIMESTAMP}.tgz" /etc/myapp/ 2>/dev/null
}
```

Adjusting Timeouts and Limits:

- **Packet Capture Duration:** Modify `timeout 60` value based on incident requirements
- **Log Collection Depth:** Change `tail -n 1000` to capture more/fewer log entries
- **Archive Size Limits:** Add size checks before compression to prevent disk exhaustion

Permission and Security Enhancements

Privilege Escalation:

Listing B.15. Sudo Integration

```
# For commands requiring elevated privileges
if sudo -l | grep -q tcpdump; then
    sudo tcpdump -i "$iface" -w "$WORKDIR/packets_${iface}_${TIMESTAMP}.pcap"
        -s 0 -U &
else
    echo "Insufficient privileges for packet capture" >
        "$WORKDIR/tcpdump_error_${TIMESTAMP}.txt"
fi
```

SSH Key Security:

- Use dedicated SSH keys for forensic operations
- Implement SSH agent forwarding for multi-hop environments
- Consider SSH certificate-based authentication for enhanced security

Error Handling and Logging

Enhanced Error Reporting:

Listing B.16. Robust Error Handling

```
execute_with_retry() {
    local command="$1"
    local max_attempts="$2"
```

```
    local attempt=1

    while [ $attempt -le $max_attempts ]; do
        if eval "$command"; then
            return 0
        else
            echo "[RETRY␣$attempt/$max_attempts]␣Command␣failed:␣$command"
            attempt=$((attempt + 1))
            sleep 2
        fi
    done
    return 1
}
```

B.1.5 Integration with Zabbix Actions

Zabbix Action Configuration:

- **Trigger Condition:** Network traffic threshold exceeded
- **Action Command:** `/usr/local/bin/cs1-collect.sh {HOST.IP} forensic`
- **Execution User:** zabbix (with appropriate SSH keys)
- **Timeout:** 300 seconds (5 minutes) to accommodate packet capture

Return Code Handling:

Listing B.17. Zabbix Integration

```
# Ensure proper exit codes for Zabbix action monitoring
if [ $? -eq 0 ]; then
    echo "[*]␣Forensic␣evidence␣collection␣completed␣successfully"
    exit 0
else
    echo "[*]␣ERROR:␣Forensic␣collection␣failed"
    exit 1
fi
```

B.1.6 Performance Considerations

Resource Usage

- **Memory:** Packet capture requires sufficient buffer space
- **Disk:** Temporary files may consume significant space during collection
- **Network:** Archive transfer impacts bandwidth during incident response
- **CPU:** Compression operations may cause temporary load spikes

Optimization Strategies

- **Parallel Collection:** Use background processes for independent operations
- **Selective Capture:** Filter packet capture by protocol or port
- **Incremental Transfer:** Stream data during collection rather than post-processing
- **Compression Tuning:** Balance compression ratio against processing time

This script represents a comprehensive approach to automated forensic evidence collection, balancing thoroughness with practical execution constraints in production environments.

B.2 Case Study 2: Unauthorized Service Detection and Evidence Collection Script

B.2.1 Script Overview

The `cs2-collect.sh` script implements specialized forensic evidence collection for unauthorized service detection scenarios. Unlike Case Study 1, this script focuses on targeted analysis of specific network services and processes, with emphasis on binary analysis, process relationships, and industrial environment context.

B.2.2 Script Architecture

Execution Model

The script employs a **parameter-driven execution model** that extracts the suspect port number from Zabbix trigger keys and performs focused forensic collection around that specific service.

Input Parameters

- `TARGET_HOST`: IP address of the victim host
- `KEY`: Zabbix item key containing port number to inspect (format: `item[port]`)
- `REMOTE_USER`: SSH username for remote access

B.2.3 Detailed Script Analysis

Phase 1: Parameter Extraction and Validation

Listing B.18. Dynamic Port Extraction

```
TARGET_HOST="$1"
KEY="$2"
REMOTE_USER="$3"

SUSPECT_PORT=$(echo "$KEY" | sed -n 's/.*\[\([0-9]\+\)\]/\1/p')

if [[ -z "$TARGET_HOST" || -z "$SUSPECT_PORT" || -z "$REMOTE_USER" ]]; then
  echo "Error: Missing parameters."
  echo "Usage: $0 <target_ip> <suspect_port> <ssh_user>"
  exit 1
fi
```

Key Design Features:

- **Dynamic Port Extraction:** Uses regex to extract port number from Zabbix item keys like `net.tcp.listen[4444]`
- **Strict Validation:** Ensures all required parameters are present before remote execution
- **Zabbix Integration:** Designed to receive Zabbix macro expansions as parameters

Phase 2: Security-Enhanced Remote Execution

Listing B.19. Secure Remote Environment

```
ssh ${REMOTE_USER}@${TARGET_HOST} bash <<ENDSSH

set -e
umask 077

TARGET_HOST="${TARGET_HOST}"
SUSPECT_PORT="${SUSPECT_PORT}"
TIMESTAMP="${TIMESTAMP}"
EVIDENCE_DIR="${EVIDENCE_DIR}"
```

Security Enhancements:

- **set -e**: Immediate script termination on any command failure
- **umask 077**: Restrictive file permissions (600) for all created evidence files
- **Variable Sanitization**: All variables are explicitly passed and validated

Phase 3: Comprehensive Metadata Collection

Listing B.20. Forensic Session Documentation

```
START_TS=$(date --iso-8601=seconds)
echo "Forensic␣Collection␣Session" > "${EVIDENCE_DIR}/${REPORT_NAME}"
echo "Target␣host:␣${TARGET_HOST}" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
echo "Session␣start:␣${START_TS}" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
echo "Suspect␣port:␣${SUSPECT_PORT}" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
echo "Run␣by␣host:␣$(hostname)" >> "${EVIDENCE_DIR}/${REPORT_NAME}"

# Tool versions
{
  echo "ss␣version:␣$(ss␣--version␣2>&1␣|␣head␣-1)"
  echo "netstat␣version:␣$(netstat␣--version␣2>&1␣|␣head␣-1)"
  echo "lsof␣version:␣$(lsof␣-v␣2>&1␣|␣head␣-1)"
  echo "ps␣version:␣$(ps␣--version␣2>&1␣|␣head␣-1)"
  echo "sha256sum␣version:␣$(sha256sum␣--version␣|␣head␣-1)"
  echo "zip␣version:␣$(zip␣-v␣|␣head␣-1)"
} >> "${EVIDENCE_DIR}/${REPORT_NAME}"
```

Enhanced Metadata Features:

- **ISO-8601 Timestamps:** Precise timing information with timezone data
- **Tool Versioning:** Documents exact versions of forensic tools used
- **Chain of Custody:** Records collection host and execution context

Phase 4: Network Service Analysis

Listing B.21. Comprehensive Network Data Collection

```
# Network data and hashes
ss -tulpn > "${EVIDENCE_DIR}/listening_ports.txt" 2>&1
sha256sum "${EVIDENCE_DIR}/listening_ports.txt" >
    "${EVIDENCE_DIR}/listening_ports.txt.sha256"
```

```
netstat -anp > "${EVIDENCE_DIR}/netstat_anp.txt" 2>&1
sha256sum "${EVIDENCE_DIR}/netstat_anp.txt" >
    "${EVIDENCE_DIR}/netstat_anp.txt.sha256"

lsof -i > "${EVIDENCE_DIR}/lsof_all_network.txt" 2>&1
sha256sum "${EVIDENCE_DIR}/lsof_all_network.txt" >
    "${EVIDENCE_DIR}/lsof_all_network.txt.sha256"
```

Multi-Tool Network Analysis:

- **ss**: Modern socket statistics with process information
- **netstat**: Legacy compatibility and detailed connection states
- **lsof**: Process-to-network mapping with file descriptor details
- **Immediate Hashing**: Each evidence file is immediately hashed for integrity

Phase 5: Advanced Process and Binary Analysis

Listing B.22. Per-Process Binary Analysis

```
while read -r line; do
  PORT=$(echo "$line" | awk '{print␣$5}' | awk -F: '{print␣$NF}')
  PIDUSER=$(echo "$line" | awk '{print␣$7}')
  if [[ -n "$PIDUSER" && "$PIDUSER" != "*" ]]; then
    PID=$(echo "$PIDUSER" | cut -d/ -f1)
    PROCNAME=$(echo "$PIDUSER" | cut -d/ -f2)
    if [[ -d "/proc/$PID" ]]; then
      EXEPATH=$(readlink -f /proc/$PID/exe 2>/dev/null)
      if [[ -n "$EXEPATH" ]]; then
        BNFILE="${EVIDENCE_DIR}/binary_${PROCNAME}_${PID}_port${PORT}.txt"
        HASHFILE="${EVIDENCE_DIR}/binary_${PROCNAME}_${PID}_port${PORT}.sha256"
        ENVFILE="${EVIDENCE_DIR}/environ_${PROCNAME}_${PID}_port${PORT}.txt"
        LSOFFILE="${EVIDENCE_DIR}/lsof_${PROCNAME}_${PID}_port${PORT}.txt"
        ls -la "$EXEPATH" > "$BNFILE" 2>&1
        sha256sum "$EXEPATH" > "$HASHFILE"
        cat /proc/$PID/environ | tr '\0' '\n' > "$ENVFILE" 2>&1
        sha256sum "$ENVFILE" > "$ENVFILE.sha256"
        lsof -p $PID > "$LSOFFILE"
        sha256sum "$LSOFFILE" > "$LSOFFILE.sha256"
      fi
    fi
  fi
done < <(ss -tulpn | tail -n +2)
```

Advanced Analysis Features:

- **Binary Fingerprinting:** SHA256 hash of each network service binary
- **Environment Analysis:** Process environment variables for execution context
- **File Descriptor Analysis:** Complete file/socket mapping per process
- **Granular Evidence:** Separate files per process-port combination

Phase 6: User Activity Forensics

Listing B.23. Multi-User History Collection

```
for userhome in /home/*/; do
  user=$(basename "$userhome")
  histfile="${userhome}.bash_history"
  if [[ -r "$histfile" ]]; then
    echo "History␣for␣user␣$user:" > "${EVIDENCE_DIR}/history_${user}.txt"
    cat "$histfile" >> "${EVIDENCE_DIR}/history_${user}.txt"
    sha256sum "${EVIDENCE_DIR}/history_${user}.txt" >
        "${EVIDENCE_DIR}/history_${user}.txt.sha256"
    echo "Collected␣history␣for␣$user." >> "${EVIDENCE_DIR}/${REPORT_NAME}"
  else
    echo "No␣readable␣history␣for␣$user." >> "${EVIDENCE_DIR}/${REPORT_NAME}"
  fi
done

if [[ -r /root/.bash_history ]]; then
  echo "History␣for␣root:" > "${EVIDENCE_DIR}/history_root.txt"
  cat /root/.bash_history >> "${EVIDENCE_DIR}/history_root.txt"
  sha256sum "${EVIDENCE_DIR}/history_root.txt" >
      "${EVIDENCE_DIR}/history_root.txt.sha256"
fi
```

Comprehensive User Activity:

- **All User Accounts:** Iterates through all home directories
- **Permission-Aware:** Only collects readable history files
- **Root Privilege Handling:** Separate collection for root account
- **Audit Trail:** Reports which users had accessible histories

Phase 7: Focused Suspect Port Analysis

Listing B.24. Targeted Port Investigation

```
echo "Collecting␣details␣for␣suspect␣port␣$SUSPECT_PORT..." >>
    "${EVIDENCE_DIR}/${REPORT_NAME}"

LSOF_FN="${EVIDENCE_DIR}/lsof_port_${SUSPECT_PORT}.txt"
lsof -i :${SUSPECT_PORT} > "$LSOF_FN"
sha256sum "$LSOF_FN" > "${LSOF_FN}.sha256"

PID=$(lsof -t -i :${SUSPECT_PORT} | head -n 1 || true)
if [[ -n "$PID" ]]; then
  echo "Suspect␣port␣${SUSPECT_PORT}␣PID:␣$PID" >>
      "${EVIDENCE_DIR}/${REPORT_NAME}"

  ps auxww | grep "^\\S*␣*$PID␣" >
      "${EVIDENCE_DIR}/ps_port_${SUSPECT_PORT}.txt"
  sha256sum "${EVIDENCE_DIR}/ps_port_${SUSPECT_PORT}.txt" >
      "${EVIDENCE_DIR}/ps_port_${SUSPECT_PORT}.txt.sha256"

  EXEPATH=$(readlink -f /proc/$PID/exe 2>/dev/null)
  if [[ -n "$EXEPATH" ]]; then
    sha256sum "$EXEPATH" > "${EVIDENCE_DIR}/exe_hash_${SUSPECT_PORT}.sha256"
```

```
    ls -la "$EXEPATH" > "${EVIDENCE_DIR}/exe_path_${SUSPECT_PORT}.txt"
    sha256sum "${EVIDENCE_DIR}/exe_path_${SUSPECT_PORT}.txt" >
        "${EVIDENCE_DIR}/exe_path_${SUSPECT_PORT}.txt.sha256"
  fi
fi
```

Targeted Analysis:

- **Port-Specific Investigation:** Focuses on the exact port that triggered the alert
- **Process Identification:** Links port to specific PID and binary
- **Binary Analysis:** Detailed metadata and hash of suspect executable
- **Process Context:** Full process information including command line arguments

Phase 8: System State Documentation

Listing B.25. System State Capture

```
pstree -p > "${EVIDENCE_DIR}/process_tree.txt"
sha256sum "${EVIDENCE_DIR}/process_tree.txt" >
    "${EVIDENCE_DIR}/process_tree.txt.sha256"

systemctl list-units --type=service --state=running >
    "${EVIDENCE_DIR}/running_services.txt"
sha256sum "${EVIDENCE_DIR}/running_services.txt" >
    "${EVIDENCE_DIR}/running_services.txt.sha256"

free -h > "${EVIDENCE_DIR}/memory_usage.txt"
sha256sum "${EVIDENCE_DIR}/memory_usage.txt" >
    "${EVIDENCE_DIR}/memory_usage.txt.sha256"

df -h > "${EVIDENCE_DIR}/disk_usage.txt"
sha256sum "${EVIDENCE_DIR}/disk_usage.txt" >
    "${EVIDENCE_DIR}/disk_usage.txt.sha256"

top -b -n 1 > "${EVIDENCE_DIR}/cpu_usage.txt"
sha256sum "${EVIDENCE_DIR}/cpu_usage.txt" >
    "${EVIDENCE_DIR}/cpu_usage.txt.sha256"
```

System Context:

- **Process Hierarchy:** Complete process tree for relationship analysis
- **Service Status:** All running systemd services
- **Resource Usage:** Memory, disk, and CPU utilization snapshots
- **Performance Impact:** Understanding system load during incident

Phase 9: File System Timeline Analysis

Listing B.26. Recent File Modifications

```
echo "Collecting␣recently␣modified␣files..." >>
    "${EVIDENCE_DIR}/${REPORT_NAME}"
find /tmp /var/tmp /opt -type f -newermt '-1␣hour' -ls >
    "${EVIDENCE_DIR}/recent_files.txt"
sha256sum "${EVIDENCE_DIR}/recent_files.txt" >
    "${EVIDENCE_DIR}/recent_files.txt.sha256"
```

Timeline Analysis:

- **Recent Activity:** Files modified within 1 hour of incident
- **Suspicious Locations:** Focuses on temporary and optional directories
- **Detailed Metadata:** Full file listing with permissions and timestamps

Phase 10: Industrial Environment Context

Listing B.27. Industrial System Logs

```
if compgen -G "/opt/industrial/logs/*" > /dev/null; then
  cat /opt/industrial/logs/* > "${EVIDENCE_DIR}/industrial_logs.txt"
  sha256sum "${EVIDENCE_DIR}/industrial_logs.txt" >
      "${EVIDENCE_DIR}/industrial_logs.txt.sha256"
  echo "Collected␣industrial␣log␣files." >> "${EVIDENCE_DIR}/${REPORT_NAME}"
else
  echo "No␣industrial␣logs␣found␣or␣directory␣empty." >>
      "${EVIDENCE_DIR}/${REPORT_NAME}"
fi
```

Industrial Context:

- **Specialized Logs:** Collection of industrial control system logs
- **Optional Collection:** Graceful handling when industrial environment not present
- **Comprehensive Logging:** Documents whether industrial components are involved

Phase 11: Hash Consolidation and Final Report

Listing B.28. Evidence Integrity Summary

```
echo "----------------------------------------" >>
    "${EVIDENCE_DIR}/${REPORT_NAME}"
echo "SHA256␣checksums␣collected:" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
find ${EVIDENCE_DIR} -type f -name '*.sha256' ! -name "${REPORT_NAME}" |
    while read -r hashfile; do
  echo "---␣${hashfile##*/}␣---" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
  cat "$hashfile" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
done

END_TS=$(date --iso-8601=seconds)
echo "----------------------------------------" >>
    "${EVIDENCE_DIR}/${REPORT_NAME}"
echo "Session␣end:␣${END_TS}" >> "${EVIDENCE_DIR}/${REPORT_NAME}"
```

Final Documentation:

- **Hash Consolidation:** All evidence hashes in central report
- **Session Timing:** Precise start and end timestamps
- **Integrity Verification:** Comprehensive hash documentation

Phase 12: Secure Transfer and Verification

Listing B.29. Secure Evidence Transfer

```
cd /tmp
zip -r "${ARCHIVE_NAME}" "$(basename␣"${EVIDENCE_DIR}")" > /dev/null 2>&1
sha256sum "${ARCHIVE_NAME}" > "${ARCHIVE_NAME}.sha256"

# Transfer and verification
scp ${REMOTE_USER}@${TARGET_HOST}:/tmp/${ARCHIVE_NAME} ${LOCAL_PATH}/
scp ${REMOTE_USER}@${TARGET_HOST}:/tmp/${ARCHIVE_NAME}.sha256 ${LOCAL_PATH}/

LOCAL_HASH=$(sha256sum ${LOCAL_PATH}/${ARCHIVE_NAME} | awk '{print␣$1}')
REMOTE_HASH=$(cat ${LOCAL_PATH}/${ARCHIVE_NAME}.sha256 | awk '{print␣$1}')

if [[ "$LOCAL_HASH" != "$REMOTE_HASH" ]]; then
  echo "Checksum␣mismatch!␣Transfer␣corrupted."
  exit 1
fi

# Cleanup
ssh ${REMOTE_USER}@${TARGET_HOST} "rm␣-rf␣${EVIDENCE_DIR}␣
    /tmp/${ARCHIVE_NAME}␣/tmp/${ARCHIVE_NAME}.sha256"
```

Transfer Security:

- **Cryptographic Verification:** SHA256 comparison between source and destination
- **Transfer Integrity:** Automatic failure detection and error reporting
- **Secure Cleanup:** Complete removal of temporary files from target system

B.2.4 Customization Guidelines

Port Extraction Customization

Alternative Key Formats:

Listing B.30. Custom Port Extraction

```
# For custom Zabbix item formats
SUSPECT_PORT=$(echo "$KEY" | sed -n 's/.*port_\([0-9]\+\).*/\1/p')

# For multiple port scenarios
SUSPECT_PORTS=$(echo "$KEY" | grep -o '[0-9]\+' | tr '\n' '␣')
```

Industrial Environment Adaptation

Additional Industrial Analysis:

Listing B.31. Extended Industrial Monitoring

```
# Modbus communication analysis
if command -v modpoll >/dev/null 2>&1; then
  modpoll -m tcp -a 1 -r 1 -c 10 -t 4 ${TARGET_HOST} >
      "${EVIDENCE_DIR}/modbus_test.txt"
  sha256sum "${EVIDENCE_DIR}/modbus_test.txt" >
      "${EVIDENCE_DIR}/modbus_test.txt.sha256"
fi
```

```
# MQTT broker analysis
if ss -tulpn | grep -q ':1883\|:8883'; then
  mosquitto_sub -h localhost -t '#' -C 100 >
      "${EVIDENCE_DIR}/mqtt_messages.txt" &
  MQTT_PID=$!
  sleep 30
  kill $MQTT_PID 2>/dev/null
  sha256sum "${EVIDENCE_DIR}/mqtt_messages.txt" >
      "${EVIDENCE_DIR}/mqtt_messages.txt.sha256"
fi
```

Enhanced Binary Analysis

Malware Detection Integration:

Listing B.32. Extended Binary Analysis

```
# Static analysis tools integration
if command -v strings >/dev/null 2>&1; then
  strings "$EXEPATH" > "${EVIDENCE_DIR}/strings_${PID}.txt"
  sha256sum "${EVIDENCE_DIR}/strings_${PID}.txt" >
      "${EVIDENCE_DIR}/strings_${PID}.txt.sha256"
fi

# File type analysis
if command -v file >/dev/null 2>&1; then
  file "$EXEPATH" > "${EVIDENCE_DIR}/filetype_${PID}.txt"
  sha256sum "${EVIDENCE_DIR}/filetype_${PID}.txt" >
      "${EVIDENCE_DIR}/filetype_${PID}.txt.sha256"
fi

# Library dependencies
if command -v ldd >/dev/null 2>&1; then
  ldd "$EXEPATH" > "${EVIDENCE_DIR}/libraries_${PID}.txt" 2>&1
  sha256sum "${EVIDENCE_DIR}/libraries_${PID}.txt" >
      "${EVIDENCE_DIR}/libraries_${PID}.txt.sha256"
fi
```

Network Traffic Analysis

Targeted Traffic Capture:

Listing B.33. Port-Specific Traffic Analysis

```
# Capture traffic specific to suspect port
if command -v tcpdump >/dev/null 2>&1; then
  timeout 60 tcpdump -i any port ${SUSPECT_PORT} -w
      "${EVIDENCE_DIR}/traffic_port_${SUSPECT_PORT}.pcap" -s 0 &
  TCPDUMP_PID=$!
  echo "Started␣packet␣capture␣for␣port␣${SUSPECT_PORT}␣(PID:␣$TCPDUMP_PID)"
  wait $TCPDUMP_PID
  sha256sum "${EVIDENCE_DIR}/traffic_port_${SUSPECT_PORT}.pcap" >
      "${EVIDENCE_DIR}/traffic_port_${SUSPECT_PORT}.pcap.sha256"
fi
```

B.2.5 Integration with Zabbix Triggers

Low-Level Discovery Integration

Zabbix Template Configuration:

- **Discovery Rule:** `industrial.services.discovery`
- **Item Prototype:** `industrial.port.listen[{#PORT}]`
- **Trigger Expression:** `{Template:industrial.port.listen[4444].last()}>0`
- **Action Command:** `/usr/local/bin/cs2-collect.sh {HOST.IP} {ITEM.KEY}}`

Advanced Trigger Conditions

Multi-Condition Triggers:

Listing B.34. Complex Trigger Logic

```
# Unauthorized service AND high CPU usage
{Template:industrial.port.listen[4444].last()}>0
and
{Template:system.cpu.util.avg(5m)}>80

# Multiple unauthorized ports
{Template:industrial.port.listen[4444].last()}>0
or
{Template:industrial.port.listen[5555].last()}>0
```

B.2.6 Performance and Security Considerations

Performance Optimization

- **Parallel Evidence Collection:** Use background processes for independent operations
- **Selective File Analysis:** Limit find operations to relevant directories
- **Compression Optimization:** Use appropriate compression levels for archive size vs. speed
- **Resource Monitoring:** Include system load checks to prevent overloading target systems

Security Enhancements

- **Privilege Isolation:** Use dedicated forensic user accounts with minimal privileges
- **Secure Communication:** Implement SSH certificate-based authentication
- **Evidence Encryption:** Add GPG encryption for sensitive evidence archives
- **Access Logging:** Log all forensic collection activities for audit trails

This script provides focused, targeted forensic collection specifically designed for unauthorized service detection scenarios, with enhanced binary analysis and industrial environment awareness.

Bibliography

[1] IBM Security, "Cost of a data breach report 2024", 2024

[2] CrowdStrike, "Crowdstrike global outage: What happened and lessons learned", 2024

[3] S. Romanosky, "A case study of the target data breach: A lesson in security failure", Journal of Information Privacy and Security, vol. 10, no. 2, 2014, pp. 1–16, DOI 10.1080/15536548.2014.1019920

[4] Microsoft Corporation, "Bitlocker drive encryption overview", 2023

[5] K. Kent, S. Chevalier, T. Grance, and H. Dang, "Guide to integrating forensic techniques into incident response", 2006

[6] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide", 2012

[7] S. Garfinkel, "Digital forensics research: The next 10 years", Digital Investigation, vol. 7, 2010, pp. S64–S73, DOI 10.1016/j.diin.2010.05.009

[8] E. Casey and G. Stellatos, "The impact of full disk encryption on digital forensics", Digital Forensics and Cyber Crime, 2011

[9] Cisco Systems, "Cisco netflow configuration guide", 2021

[10] CERT NetSA Team, "Yet another flowmeter (yaf)", 2023

[11] T. Z. Project, "Zeek network security monitor", 2023, `https://zeek.org/`

[12] A. Chuvakin, K. Schmidt, and C. Phillips, "Security information and event management (siem) implementation", McGraw-Hill Education, 2013, ISBN: 9780071742382

[13] B. Turnbull, B. Irwin, and W. A. Labuschagne, "Network forensics: An analysis of techniques, tools, and trends", The Journal of Digital Forensics, Security and Law, vol. 13, no. 4, 2018, pp. 15–36, DOI 10.15394/jdfsl.2018.1535

[14] Snort Project, "Snort users manual", 2023

[15] OISF, "Suricata - open source ids/ips/nsm engine", 2024

[16] A. Chuvakin and K. Scarfone, "Security information and event management (siem) implementation", NIST Special Publication, no. 800-92, 2007

[17] IBM Security, "Ibm qradar siem overview", 2023

[18] Elastic Co., "Elastic stack documentation", 2024

[19] A. Alazab and Y. Al-Nashif, "Soar: Security orchestration, automation and response - techniques and applications", Journal of Cybersecurity and Privacy, 2021

[20] Zabbix Team, "Security monitoring with zabbix", 2023

[21] X. Li, Y. Zhang, and H. Chen, "Autoforensics: Real-time automated forensic evidence collection architecture", Proceedings of IEEE Symposium on Security and Privacy Workshops, 2020, pp. 285–294, DOI 10.1109/SPW50608.2020.00053

[22] F. Marturana, S. Tacconi, and A. Merlo, "A framework for cloud forensic readiness in organizations", Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER), 2011, pp. 271–278, DOI 10.5220/0003391102710278

[23] E. Salfati and M. Pease, "Digital forensics and incident response (dfir) framework for operational technology (ot)", NIST Interagency Report NIST IR 8428, National Institute of Standards and Technology, June 2022. This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8428

[24] G. Michelet, F. Breitinger, and G. Horsman, "Automation for digital forensics: Towards a definition for the community", Forensic Science International: Digital Investigation, vol. 349, August 2023, p. 111769, DOI 10.1016/j.forsciint.2023.111769. Creative Commons Attribution Non-Commercial No Derivatives License (CC:BY:NC:ND 4.0)

[25] C. B. Cunha and D. Cid, "Ossec: Host-based intrusion detection and log analysis", Technical Whitepaper, 2008. Open Source HIDS Platform Documentation

[26] J. Manzoor, A. Waleed, A. F. Jamali, and A. Masood, "Cybersecurity on a budget: Evaluating security and performance of open-source siem solutions for smes", PLOS ONE, vol. 19, March 2024, p. e0301183, DOI 10.1371/journal.pone.0301183

[27] Jumiaty and B. Soewito, "Protecting applications with wazuh and thehive: Siem and threat intelligence implementation", Technical Report 9, Bina Nusantara University, June 2024. Computer Science Department, BINUS Graduate Program

[28] A. Ammar, M. Karim, F. Alshami, and S. Hasan, "A survey of security information and event management (siem) and security orchestration, automation and response (soar) platforms", IEEE Access, vol. 9, 2021, pp. 154109–154121, DOI 10.1109/ACCESS.2021.3127685

[29] A. Miller and A. Singh, "Chain of custody and evidence integrity verification using blockchain technology", Proceedings of the 19th International Conference on Cyber Warfare and Security (ICCWS), March 2024, p. 2025, DOI 10.34190/iccws.19.1.2025. Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

[30] G. Kumar, R. Saha, M. K. Rai, R. Thomas, and S.-J. Lim, "Internet-of-forensic (iof): A blockchain based digital forensics framework for iot applications", Future Generation Computer Systems, vol. 120, July 2021, pp. 13–25, DOI 10.1016/j.future.2021.02.016. Accepted author manuscript - cited as 2025 reference for latest developments

[31] J. Wayman, K. Scarfone, P. Mell, and T. Grance, "Guidelines on mobile device forensics", Tech. Rep. NIST Special Publication 800-101 Revision 1, National Institute of Standards and Technology (NIST), Gaithersburg, MD, May 2014

[32] C. Perera, C. H. Liu, K. Jayawardena, and G. Min, "Anomaly detection and prediction for smart cities: A survey", IEEE Communications Surveys & Tutorials, vol. 21, no. 1, 2019, pp. 779–829

[33] L. Wang, X. Zhao, Y. Tang, and Y. Jin, "A machine learning-based forensic framework for intelligent intrusion detection and investigation", Digital Investigation, vol. 44, 2023, pp. 301–312

[34] D. Marzano, I. N. Fovino, and A. Carcano, "An architectural approach for real-time forensics in industrial control systems", Computers & Security, vol. 100, 2021, p. 102086

[35] A. Pal and M. K. Rogers, "Building reliable timelines in digital investigations: Challenges and opportunities", 2019 IEEE Security and Forensics, 2019

[36] S. G. Brester, "Fail2ban github repository", `https://github.com/fail2ban/fail2ban`

[37] Zabbix SIA, "Zabbix documentation: Escalations and action operations", 2025