



**Università
di Genova**

DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Study and Application of the YARA Tool for Advanced Threat Detection in Integrated IT/OT Network Infrastructures

Alessio Simeoni

Master Thesis

Università di Genova, DIBRIS

Via Opera Pia, 13 16145 Genova, Italy

<https://www.dibris.unige.it/>



**Università
di Genova**

MSc Computer Engineering

Software and Computing Platforms Curriculum

**Study and Application of the YARA Tool for Advanced
Threat Detection in Integrated IT/OT Network
Infrastructures**

Alessio Simeoni

Advisor: Alessandro Armando

October, 2026

1. Operational and Technological Context	5
1.1 The Evolution of Industrial Systems and IT/OT Convergence	5
1.2 Reference Architectures: The PERA Model and SCADA Systems	6
1.3 Threat Hunting and Malware Analysis in IACS Environments: The Role of YARA	8
1.4 Objectives and Methodology of the Thesis	8
2. Detection and Analysis Technologies	10
2.1 Monitoring Methodologies in OT Infrastructures (IDS, IPS, and Current Limitations)	10
2.1.1 The Role and Mechanism of IDS in OT/ICS	10
2.1.2 IDS vs. IPS: A Critical Distinction in the OT World	11
2.1.3 Core Challenges of IDS/IPS in the OT Environment	12
2.1.4 Current Limitations and the Evolution of Detection	13
2.1.5 The Market Landscape: Commercial Solutions for OT Protection	13
2.2 YARA: Architecture, Operation, and Rule Syntax	15
2.2.1 What is YARA?	15
2.2.2 The Anatomy of a YARA Rule	16
2.2.3 Rule Syntax and Examples	16
2.2.4 Advanced YARA Capabilities	18
2.2.5 Operational Use Cases in Cybersecurity	19
2.2.6 The YARA Ecosystem and Supporting Tooling	20
2.3 The Open Source Ecosystem for Malware Analysis	21
2.3.1 The Challenge of Payload Acquisition	21
2.3.2 MalwareBazaar: The Open Malware Repository	22
2.3.3 VirusTotal: Multiscanning and Global Threat Intelligence	23
2.3.4 Threat Intelligence and OT-Specific Analysis (Kaspersky Lab & ICS CERT)	24
2.3.5 The Analytical Pipeline: From Sample to YARA Rule	24
2.4 DFIR Tools: Introduction to Velociraptor	25
2.4.1 What is Velociraptor?	26
2.4.2 Architectural Deployment Models	26
2.4.3 The Core Differentiator: Velociraptor Query Language (VQL)	27
2.4.4 Velociraptor Artifacts: Encapsulating Forensic Knowledge	27
2.4.5 The Velociraptor Ecosystem and Integration	28
2.4.6 YARA Integration in Velociraptor	29
2.4.7 Best Practices for YARA Hunting in OT Environments	30
3: Design and Setup of the Experimental Environment	30
3.1. The "Digital Twin" Concept: Virtualization and Storage Infrastructure	31
3.2. Network Architecture and IT/OT Segregation	31
3.3. Emulating the Industrial Process: Active Directory, PLCs, and SCADA HMI	32
3.4. Attack and Defense Tooling Setup (Kali Linux and Velociraptor EDR)	34
3.5. Vulnerability Analysis of the Modbus TCP Protocol	35
3.5.1 Structural Anatomy of a Modbus TCP Packet	35
3.5.2 Inherited Security Vulnerabilities	36
4. Advanced Threat Modeling: The "Stuxnet-like" Attack Scenario	37
4.1. Initial Compromise and Lateral Movement (From DMZ to Active Directory)	37
4.2. Privilege Escalation via Domain Controller Compromise	38

4.2.1 Exploiting CVE-2020-1472 (ZeroLogon)	39
4.2.2 NTDS Dumping and Operational Security (OpSec)	39
4.2.3 Overpass-the-Hash (Kerberos Ticket Generation)	40
4.3. The False Data Injection (FDI) Attack: Modbus Manipulation via NetfilterQueue (MITM)	40
4.3.1 Sideloaded Dependencies in an Air-Gapped Environment	41
4.3.2 Host-Based Traffic Interception (IPTables)	41
4.3.3 Packet Mangling and False Data Injection	41
4.4. HMI Spoofing: Deceiving the SCADA Operator	42
5. Detection Strategies: From Known Malware to Semantic Memory Hunting	43
5.1 Evaluating Network-Based Detection on Known Malware	43
5.1.1 Acquisition and Reverse Engineering of ICS Payloads.	44
5.1.2 The Evasion Problem: Limitations of Network-Based Inspection.	44
5.2 Shifting the Paradigm: EDRs and Endpoint File Scanning	46
5.2.1 Bypassing Network Limitations	46
5.2.2 Implementation via Velociraptor	47
5.2.3 The Inherent Limitation of Signature-Based Detection	47
5.3 The Fileless Threat: Detecting FDI in SCADA Memory	48
5.3.1 The State of the Art and the "Sensory Blindness" Dilemma.	48
5.3.2 The Innovation: Semantic Memory Hunting.	48
5.4 Reverse Engineering the JVM Heap (Tomcat/ScadaBR)	49
5.4.1 Memory Dumping and Identification	50
5.4.2 Overcoming the Garbage Collection "Phantom" Matches	50
5.4.3 The Semantic Signature Structure	51
5.5 Developing the Custom VQL Artifact: Correlating RAM Data with PLC States	51
5.5.1 Scalability and Batch Execution Strategy	52
5.5.2 Out-of-Band Polling and Dynamic Signature Generation	52
5.5.3 The Custom VQL Artifact Execution	53
6. Experimental Results and Performance Evaluation	54
6.1 Comparative Efficacy on Known ICS Malware Families	54
6.2 Validation of the Semantic Memory Hunting Mechanism	57
6.2.1 Baseline Verification and False Positive Rate (FPR)	57
6.2.2 FDI Attack Detection Validation	57
6.3 System Impact Analysis on ICS Endpoints	58
6.3.1 Experimental Constraints and Baseline Configuration	58
6.3.2 Analysis of Memory Footprint (RAM)	59
6.3.3 Analysis of Computational Overhead (CPU) and Throttling	60
7. Limitations and Conclusions	62
7.1 Limitations of the Experimental Infrastructure and the Proposed Detection Artifact	62
7.2 Conclusions on the Work Performed	62
7.3 Future Developments	64
8. References	64

1. Operational and Technological Context

1.1 The Evolution of Industrial Systems and IT/OT Convergence

Industrial systems boast a decades-long history of automation, characterized by the integration of hardware and software technologies layered over time. The proper functioning of these architectures, known as Industrial Automation and Control Systems (IACS), plays a crucial role in contemporary society, ensuring large-scale goods production and, above all, the provision of essential services for modern life (such as energy, water, and gas distribution).

Historically, industrial ecosystems have been primarily characterized by their logical and physical isolation from external networks, a paradigm known as air-gapping. Systems belonging to the Operational Technology (OT) domain—defined as the hardware and software that detects or causes a change through the direct monitoring and/or control of physical devices, processes, and enterprise events—were conceived and developed under the assumption of operating in closed networks, disconnected from the Internet or traditional corporate networks.

Today, however, this scenario is undergoing a drastic and irreversible change. Driven by the Industry 4.0 and Industrial Internet of Things (IIoT) paradigms, the industrial sector is experiencing a profound convergence between the Information Technology (IT) and OT domains. Companies are interconnecting industrial systems with remote services, cloud platforms, and enterprise networks to leverage the full potential of connected facilities: predictive maintenance, real-time production optimization, and advanced telemetry.

While this convergence unleashes enormous business potential on one hand, it poses critical challenges on the other regarding adaptation, integration, and, above all, cybersecurity. Whereas IT priorities follow the classic CIA triad paradigm (Confidentiality, Integrity, Availability), OT priorities are inverted (Availability, Integrity, Confidentiality), with the addition of the fundamental requirement of physical safety. A disruption in the IT domain causes a service outage or financial loss; a disruption or tampering in the OT domain can cause physical damage to equipment and the environment, or even the loss of human life. This convergence equally involves legacy systems (which are often obsolete, lack built-in cryptographic or authentication features, and are impossible to update without plant downtime) and newly installed systems.

1.2 Reference Architectures: The PERA Model and SCADA Systems

To fully comprehend the vulnerabilities introduced by the IT/OT convergence, it is necessary to contextualize industrial systems within standardized reference architectures, such as the Purdue Enterprise Reference Architecture (PERA).

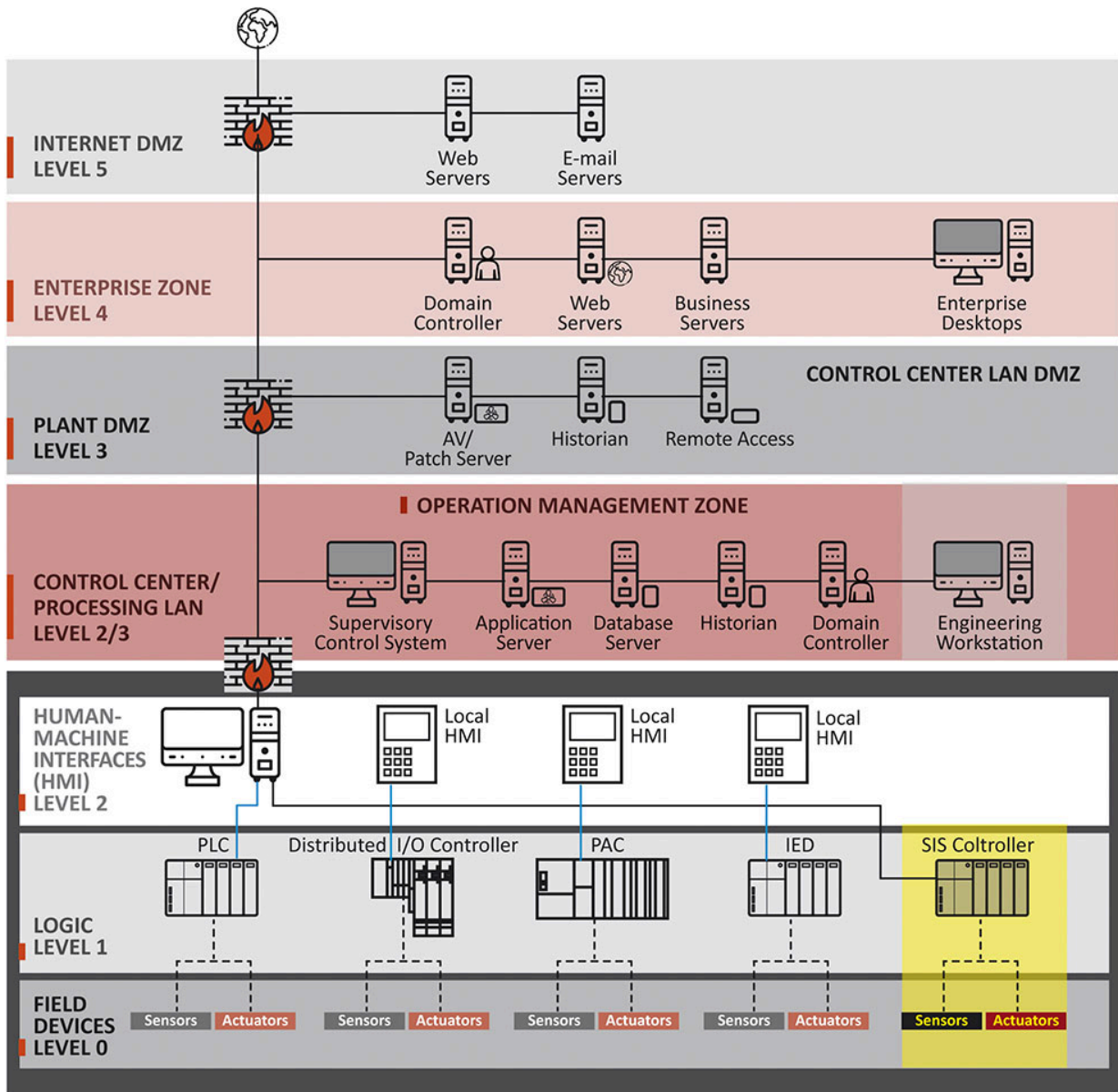


Figure 1: Purdue Model.

The Purdue model organizes the corporate and industrial network infrastructure into strict hierarchical tiers (from Level 0, the field sensors and actuators, up to Level 5, the enterprise network), defining the security zones and the permitted communication flows between them.

Within the operational levels of this model (specifically Levels 1 and 2, as depicted in Figure 1) reside SCADA (Supervisory Control And Data Acquisition) systems. These systems represent the beating heart of industrial monitoring, collecting real-time data from field devices (such as Programmable Logic Controllers - PLCs at Level 1) and presenting it to operators through graphical interfaces known as HMIs (Human Machine Interfaces) at Level 2.

The PERA model was traditionally adopted to enforce network segmentation and mitigate the broad risks associated with IT/OT convergence by isolating critical industrial assets. However, while structural frameworks like PERA provide a foundational defense-in-depth strategy, they are not sufficient on their own to guarantee security.

If a network is inadequately segmented, or if a sophisticated cyber threat successfully breaches the external perimeters (IT Levels), the attack can propagate laterally and vertically until it reaches the critical OT levels. Once inside these operational zones, the inherent vulnerabilities of legacy industrial systems are fully exposed.

This risk is deeply exacerbated by the use of industrial protocols that are often obsolete or fundamentally vulnerable. A prime example is the implementation of the Modbus stack over TCP/IP, a widely deployed protocol that was originally designed without built-in security features (such as authentication or payload encryption), making it an easy target for Man-in-the-Middle (MITM) attacks, Replay Attacks, or command injection once the perimeter is breached.

To address these challenges, the regulatory landscape has developed rigorous guidelines. A primary reference in this domain is the IEC 62443 standard, the international normative suite specifically designed for the cybersecurity of IACS systems. This standard promotes a defense-in-depth approach based on network segmentation into Zones and Conduits, establishing escalating security requirements based on the criticality of the industrial components.

However, while IEC 62443 provides a comprehensive framework and a robust set of security controls, it ultimately delegates the specific risk assessment and the practical implementation of these security mechanisms to individual organizations. Within this context, this thesis focuses on the operational implementation of active defense mechanisms. Specifically, it bridges the gap between theoretical compliance and practical security by exploring how advanced Endpoint Detection and Response (EDR) tools and custom YARA rules can proactively detect sophisticated threats—such as False Data Injection (FDI)—that manage to bypass the structural mitigations proposed by the standard.

1.3 Threat Hunting and Malware Analysis in IACS Environments: The Role of YARA

This thesis work, supported by the expertise and the IoT/IIoT infrastructure of Relatech S.p.A.—a company that has been engaged for years in developing security projects for industrial automation systems—focuses on the advanced experimentation of Threat Hunting techniques within the OT domain, with a particular emphasis on the use of the YARA tool.

It has become evident that traditional perimeter defense measures are insufficient to stop sophisticated threats such as APTs (Advanced Persistent Threats). APT attacks in SCADA environments are designed to evade detection, remaining dormant within the network for extended periods of time to study industrial processes and exfiltrate sensitive data before executing a destructive attack. Therefore, it is essential not only to promptly identify and block an intrusion reported by IDS/IPS (Intrusion Detection/Prevention System) solutions, but also to acquire and analyze the malware sample to launch a proactive hunt (Threat Hunting) across the infrastructure. The objective is to verify whether the malicious agent has spread to other devices and to contain its reach.

This context introduces YARA, an open-source tool initially developed in the IT domain to assist security researchers in identifying and classifying malicious software samples. Today, however, its application extends far beyond laboratory research; YARA has become a de facto industry standard widely deployed in production environments. It is actively utilized by Security Operations Centers (SOCs) and natively integrated into modern enterprise security platforms—such as EDRs and network monitoring tools—for real-time threat detection and continuous incident response. YARA enables the creation of descriptions for malware families based on textual or binary patterns (strings, hexadecimal sequences, file metadata). Building a customized database of YARA rules allows for the scanning of the infrastructure in search of specific Indicators of Compromise (IoCs). Although YARA is a de facto standard in IT, its application in the OT environment is still maturing, even though it is garnering strong interest and is progressively being integrated into leading commercial tools for industrial monitoring (such as the solutions offered by Nozomi Networks and Kaspersky Industrial Cyber Security).

1.4 Objectives and Methodology of the Thesis

Based on the described technological context, the primary objective of this thesis work is the analysis and simulation of attack vectors in OT environments, in order to develop detection methodologies based on custom rules.

The research activities, conducted in a laboratory setting, will follow a methodological approach divided into several operational phases:

- **Architectural and Protocol Analysis:** An in-depth study of the most widely used industrial protocols for supervision and control within an infrastructure structured according to the PERA model, with specific focus on the structural and cryptographic limitations of Modbus TCP/IP (addressed in Chapter 3).
- **Offensive Threat Modeling (Red Teaming):** Simulation of an Advanced Persistent Threat (APT) multi-stage attack lifecycle targeting the industrial control pipeline. This involves initial access via DMZ exploitation, privilege escalation and domain compromise via Zerologon and Kerberos ticket manipulation, and local host-based False Data Injection (FDI) via NetfilterQueue to achieve sensory blindness on the SCADA HMI (detailed in Chapter 4).
- **Malware Sample Acquisition and Analysis Pipeline:** The theoretical and operational foundation for acquiring known malware samples from public intelligence repositories (such as MalwareBazaar), extracting static and behavioral indicators, and constructing specialized YARA detection rules, also exploring the use of Large Language Models (LLMs) to assist in the automated generation and optimization of these signatures (discussed in Chapter 2 and executed in Chapter 5).
- **Comparative Detection Benchmarking on Known Malware:** Implementation and evaluation of YARA rules against known industrial and IT malware samples across heterogeneous detection tiers, explicitly comparing the capabilities and limitations of passive Network Intrusion Detection Systems (Nozomi Networks, YAIDS) versus host-based execution platforms (YARA+Python, Velociraptor) (detailed in Chapter 5 and analyzed in Chapter 6).
- **Design of Semantic Memory Hunting (Blue Teaming):** Transcending static malware detection to confront fileless process tampering. This phase focuses on reverse-engineering the SCADA application's runtime state in volatile memory (JVM heap) and engineering a custom Velociraptor VQL artifact capable of correlating physical PLC states with HMI memory addresses to unmask FDI attacks (developed in Chapter 5).
- **Experimental Validation and Performance Profiling:** Quantitative evaluation of the proposed detection architectures, assessing detection rates across malware families, false-positive mitigation, and the computational footprint (CPU and RAM overhead measured via system profiling tools) of continuous memory scanning on industrial endpoints (evaluated in Chapter 6), concluding with architectural hardening recommendations and future research directions (presented in Chapter 7).

Through this process, the thesis demonstrates how analysis tools developed for Information Technology can be successfully adapted to protect the operational resilience of Operational Technology.

2. Detection and Analysis Technologies

2.1 Monitoring Methodologies in OT Infrastructures (IDS, IPS, and Current Limitations)

The cybersecurity of Operational Technology (OT) and Industrial Control Systems (ICS) relies fundamentally on robust network monitoring methodologies. As industrial environments transition from isolated, air-gapped architectures to highly interconnected ecosystems, the ability to monitor network traffic and detect anomalous behavior has become a critical operational requirement. At the core of these defensive strategies are Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), which act as the digital watchdogs of the industrial network, designed to identify, analyze, and mitigate malicious activities before they can compromise critical physical processes.

2.1.1 The Role and Mechanism of IDS in OT/ICS

Intrusion Detection Systems (IDS) monitor network traffic by performing deep inspection of data packets, analyzing source and destination IP addresses, communication ports, and payload contents. When an IDS detects a new, unauthorized, or suspicious pattern, it generates an alert for the Security Operations Center (SOC) or the local engineering team.

In an OT environment, IDS solutions are typically deployed out-of-band using Switched Port Analyzer (SPAN) ports, Test Access Points (TAPs), or through integrated sensors embedded directly within OT network switches. This architecture ensures that the monitoring process remains entirely passive and non-intrusive.

Advantages of IDS in Operational Technology:

- **High Efficacy in Static Networks:** OT networks are inherently deterministic. Unlike IT environments where users constantly browse different websites or download arbitrary files, OT devices (such as PLCs and HMIs) communicate with specific peers using predictable patterns and fixed intervals. IDS solutions excel in these static environments by easily establishing a baseline of "normal" behavior.

- **Detection of Known Attack Patterns:** Traditional IDS relies on signature-based detection, which is highly effective at immediately flagging known malware, exploits, or default credential usage.
- **Non-Intrusive Nature:** Because an IDS operates passively on a copy of the network traffic, it introduces zero latency and completely eliminates the risk of accidentally blocking legitimate industrial commands.

2.1.2 IDS vs. IPS: A Critical Distinction in the OT World

While often mentioned together, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) serve distinct operational purposes, and their deployment in an OT context requires careful consideration.

- **Intrusion Detection System (IDS):** As an out-of-band, passive monitoring tool, an IDS watches a mirrored copy of network traffic. It logs activity, collects forensic data, and generates alerts, but it does not actively block traffic. This characteristic is ideal for highly sensitive OT environments located in the lower levels of the Purdue Model (Levels 0-2). In these zones, a false positive—such as erroneously blocking a legitimate "valve open" or "emergency stop" command—could lead to physical damage, environmental hazards, or emergency plant shutdowns.
- **Intrusion Prevention System (IPS):** An IPS is an active control mechanism deployed in-line with the network traffic. It inspects packets in real-time and, upon detecting a known threat or policy violation, instantly takes preventative action. This can include dropping malicious packets, blocking the source IP, or terminating the TCP session. While an IPS offers maximum protection by neutralizing threats before they reach their target, its in-line placement demands extreme precision.

In modern industrial architectures, the most effective security posture is often a hybrid approach. This involves deploying an OT-aware IDS for maximum visibility and safety within the deeper control networks (Levels 0-2), combined with a carefully configured, OT-aware IPS deployed at the IT/OT network boundary or Industrial Demilitarized Zone (IDMZ) (Levels 3-3.5) to block external threats from entering the control environment.

2.1.3 Core Challenges of IDS/IPS in the OT Environment

Implementing traditional IT-centric IDS/IPS solutions in an industrial setting is a recipe for operational failure. OT networks present unique, structural challenges that specialized industrial detection systems must overcome:

- **Deep Packet Inspection (DPI) for Industrial Protocols:** While IT traffic relies on standard protocols like HTTP, HTTPS, or SMTP, OT networks utilize specialized, and sometimes proprietary, protocols such as Modbus/TCP, Ethernet/IP, DNP3, IEC 61850, and OPC UA. An effective industrial IDS/IPS must possess a native, deep understanding of these protocols (Deep Packet Inspection) to validate the legitimacy of industrial commands (e.g., distinguishing a standard read request from an unauthorized firmware write command) and detect protocol misuse.
- **Legacy Systems and "Unpatchable" Assets:** Many ICS devices currently in operation are decades old, lack built-in security features, and run on obsolete or proprietary operating systems. Furthermore, patching a PLC often requires a planned plant shutdown, which can take months to schedule. Since host-based security agents cannot be installed on these devices, the network IDS/IPS must shoulder the burden of detecting attacks targeting these unpatchable vulnerabilities.
- **Prioritization of Availability and Integrity:** The classic IT security triad prioritizes Confidentiality. In OT, the paradigm is inverted: Availability and Integrity (AIC) are paramount, alongside physical Safety. A security tool that shuts down a power grid turbine due to a false positive is fundamentally failing its core mission. Therefore, OT detection systems must maintain near-zero false-positive rates.
- **Static and Asymmetric Traffic Patterns:** OT traffic follows highly predictable, repetitive patterns (e.g., a SCADA server polling a RTU every 5 seconds). This predictability is the foundational element that allows modern anomaly-based detection engines to function effectively in industrial settings.
- **Passive Asset Discovery:** Active network scanning (commonly used in IT via tools like Nmap) can easily crash a legacy PLC network stack, causing a Denial of Service (DoS). Consequently, industrial IDS solutions must rely on purely passive network monitoring to build comprehensive, up-to-date inventories of all industrial assets, their firmware versions, and their communication baselines.

2.1.4 Current Limitations and the Evolution of Detection

Despite their critical role, traditional IDS deployments face significant limitations. Foremost among these is the reliance on signature-based detection, which is increasingly considered obsolete against advanced threats. Signatures can only identify known malware; they are entirely blind to Zero-Day exploits or customized Advanced Persistent Threats (APTs) specifically engineered to target a particular facility. Furthermore, standard IDS solutions often generate a high volume of false positives and lack the deep operational context required to differentiate between a cyberattack and a legitimate engineering misconfiguration.

To address these limitations, the industry is shifting towards Anomaly Detection Systems powered by Machine Learning (ML) and Artificial Intelligence (AI). By baselining the static OT environment, these advanced systems can detect subtle deviations in network flows or PLC logic changes that signatures would miss.

Furthermore, it is widely recognized that an IDS alone is not a silver bullet. Securing OT environments requires a strict defense-in-depth approach, combining IDS with Next-Generation Firewalls (NGFW), Security Information and Event Management (SIEM) systems, and regular penetration testing. In this context, Open-Source engines like SNORT, Suricata, Zeek, and notably YARA, are providing robust, highly customizable solutions for OT network monitoring and advanced threat hunting, bridging the gap between static detection and proactive defense.

2.1.5 The Market Landscape: Commercial Solutions for OT Protection

The market for industrial-specific IDS/IPS and Network Detection and Response (NDR) has rapidly matured, transitioning from niche startups to enterprise-grade platforms. The leading solutions today excel at addressing the OT-specific challenges mentioned above. A review of the state-of-the-art commercial landscape highlights the following top contenders:

1. Nozomi Networks (Vantage / Guardian): Recognized as a market leader, Nozomi provides deep-dive visibility and AI-driven anomaly detection. Its core strength lies in its Self-Learning Behavioral Anomaly Detection, which builds a baseline of "normal" industrial behavior and flags subtle deviations, making it highly effective against zero-day threats. It is managed centrally via Vantage for hyper-scale environments.

2. Claroty (Continuous Threat Detection - CTD): Claroty excels in generating detailed, multi-dimensional asset inventories and mapping industrial communications. It focuses heavily on IT-OT integration, seamlessly feeding data into enterprise SIEMs and firewalls, and provides unique capabilities for securing remote access to OT assets.
3. Palo Alto Networks (NGFW with OT Security): Applying industry-leading prevention to the OT boundary, Palo Alto extends its App-ID capabilities to include industrial protocols. This allows for strict policy-based segmentation at the IT/OT demarcation point (IDMZ), backed by its integrated WildFire cloud-based threat intelligence.
4. Cisco Cyber Vision: Cisco embeds visibility directly within the network infrastructure by integrating industrial sensors into its networking gear (IE switches and routers). This simplifies deployment and leverages Cisco ISE for granular policy enforcement and Talos for up-to-the-minute signature updates.
5. Dragos Platform: Built by ICS incident responders, Dragos focuses heavily on threat intelligence-driven hunting. Its "Neighborhood Watch" system translates intelligence on specific OT threat groups (e.g., EKANS, Electrum) into actionable detections, heavily utilizing the MITRE ATT&CK for ICS framework.
6. Forescout SilentDefense (formerly SecurityMatters): A pure-play passive IDS designed to be highly non-intrusive. It is highly valued in legacy "brownfield" sites for its detailed protocol parsing, compliance reporting, and anomaly detection tuned specifically for control loop disruptions.
7. Trend Micro TippingPoint (NGIPS): Focusing on high-performance, in-line prevention, TippingPoint incorporates industrial protocol awareness for high-speed network segments. It leverages the Zero Day Initiative (ZDI) to provide crucial "virtual patching" for vulnerabilities until vendor patches can be deployed.
8. Fortinet FortiGate (with FortiGuard OT Security Services): Offering a cost-effective, scaled-down perimeter protection model, Fortinet integrates signature-based industrial IDS/IPS capabilities into its broader Security Fabric. It is particularly suitable for highly distributed environments like utility substations or pipelines.
9. Radiflow iSID: Tailored for small to medium-sized OT networks, Radiflow provides behavioral baselining and risk-assessment tools that help constrained teams prioritize remediation efforts based on network topology and asset criticality.
10. SentinelOne (Vigilance for ICS): Representing the shift from EDR to Extended Detection and Response (XDR), SentinelOne integrates passive network monitoring with host-based

protection on OT endpoints (such as Windows-based HMIs or engineering workstations). This provides a powerful, contextual view of the attack chain across the IT-OT boundary.

2.2 YARA: Architecture, Operation, and Rule Syntax

In the contemporary cybersecurity landscape, threat actors are continuously evolving their techniques. To evade detection, attackers routinely alter file names, obfuscate payloads, employ polymorphic code, and recompile binaries to change their cryptographic hashes. Traditional antivirus engines, which rely heavily on static signature-based detection (such as MD5 or SHA-256 hashes), often fail to keep pace with these rapidly shifting threats.

To bridge this gap, security analysts and researchers require a mechanism that detects the fundamental characteristics and behaviors of malicious software, rather than relying on superficial file attributes. This requirement is fulfilled by YARA, a lightweight, highly efficient, and incredibly powerful pattern-matching tool that has become a cornerstone of modern digital forensics, malware research, and threat hunting.

2.2.1 What is YARA?

YARA, an acronym originally standing for "Yet Another Ridiculous Acronym", is a rule-based detection engine designed to identify and classify malware samples. Developed initially by Victor Alvarez at VirusTotal, YARA has transcended its origins to become the "pattern-matching Swiss Army knife" for security professionals worldwide.

Instead of searching for an exact file match, YARA allows analysts to create descriptive rules—essentially search templates—that define how a malicious file "looks" internally. It operates by evaluating files or memory dumps against user-defined textual or binary patterns, utilizing a robust boolean logic system.

YARA is a multi-platform tool, capable of running natively on Windows, Linux, and macOS. It can be utilized directly via a command-line interface (CLI) or integrated seamlessly into custom Python automation scripts using the `yara-python` extension. Today, it is an industry-standard technology embedded into the platforms of major cybersecurity vendors, including Cisco Talos, Kaspersky, FireEye, Nozomi Networks, and Claroty, demonstrating its applicability in both IT and Operational Technology (OT) environments.

2.2.2 The Anatomy of a YARA Rule

At its core, YARA analyzes files using rules written by security professionals. Each rule operates as a distinct programmatic function, describing a set of features that define a specific threat. A standard YARA rule is composed of a designated name and up to three primary sections: meta, strings, and condition.

1. The Meta Section

The meta section is utilized to store metadata and contextual information about the rule. While this section does not affect the detection logic or the execution of the rule, it is crucial for organizational purposes, especially when managing thousands of rules in a Security Operations Center (SOC) or a threat intelligence repository. Analysts use this section to document the rule's author, description, threat level, target industry, or reference links.

2. The Strings Section

The strings section is where the core artifacts or patterns are declared. Because malware is ultimately a software application, it utilizes strings to store data (such as IP addresses, Bitcoin wallet addresses, registry keys, or embedded commands). YARA allows the definition of various string types:

- Text (ASCII/Wide): Standard alphanumeric strings enclosed in double quotes.
- Hexadecimal Strings: Byte sequences enclosed in curly brackets { }, ideal for detecting specific opcodes or binary signatures.
- Regular Expressions (Regex): Patterns enclosed in forward slashes / /, allowing for flexible matching of dynamic data, such as shifting URLs or obfuscated scripts.

3. The Condition Section

The condition section is the logical heart of the YARA rule. It dictates the boolean expression that must be satisfied for the rule to trigger a positive match. This section leverages the variables defined in the strings section and evaluates them using operators such as and, or, not, as well as mathematical operators.

2.2.3 Rule Syntax and Examples

To understand how these components interact, it is useful to examine practical examples, ranging from basic string detection to complex malware classification.

Example 1: Basic String Detection The simplest implementation of a YARA rule is to search for a specific, readable string within a file.

```
rule helloworld_checker {  
  meta:  
    description = "Detects the presence of the Hello World string"  
    author = "Security Analyst"  
  strings:  
    $hello_world = "Hello world!"  
    $hello_world_lowercase = "hello world"  
    $hello_world_uppercase = "HELLO WORLD"  
  condition:  
    any of them  
}
```

In this example, the rule `helloworld_checker` declares three text variables. The condition `any of them` dictates that if the scanning engine encounters even one of these strings, the file will be flagged.

Example 2: Binary and Hexadecimal Pattern Matching Malware classification often requires looking at the compiled binary level. The following rule, `silent_banker`, demonstrates how YARA identifies specific byte sequences representing malicious code execution:

```
rule silent_banker : banker {  
  meta:  
    description = "Identifies the Silent Banker trojan"  
    threat_level = 3  
    in_the_wild = true  
  strings:  
    $a = { 6A 40 68 00 30 00 00 6A 14 8D 91 }  
    $b = { 8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9 }  
    $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"  
  condition:  
    $a or $b or $c  
}
```

This rule instructs the engine to flag any file containing specific hexadecimal byte patterns (\$a or \$b) or a unique hardcoded encryption key (\$c).

2.2.4 Advanced YARA Capabilities

While simple string matching is highly effective, modern malware requires more sophisticated detection techniques. YARA provides advanced capabilities that elevate it beyond a simple search tool:

- **String Modifiers:** YARA allows modifiers to be appended to strings. For example, adding the `nocase` modifier to a string makes the search case-insensitive, drastically reducing the number of variables an analyst needs to declare.
- **Counting and Iteration:** Analysts can specify how many times a string must appear. A condition such as `#a == 6` and `#b > 10` requires string \$a to appear exactly six times and string \$b more than ten times.
- **File Size Restrictions:** To optimize scanning performance and reduce false positives, conditions can evaluate the file size (e.g., `filesize > 200KB`).
- **Modules:** YARA's true power in malware reverse engineering comes from its modular architecture. By importing modules like `pe` (Portable Executable) or `elf` (Executable and Linkable Format), analysts can inspect the structural components of a compiled binary.

For instance, the following logic looks for a Windows executable importing a specific API commonly abused by malware:

None

```
import "pe"

rule Suspicious_API_Call {
    condition:
        pe.imports("kernel32.dll", "VirtualAlloc")
}
```

This rule detects binaries that call `VirtualAlloc`, a common application programming interface (API) function frequently utilized by malware to allocate memory space for unpacking hidden payloads.

2.2.5 Operational Use Cases in Cybersecurity

In modern cybersecurity operations, YARA sits perfectly between static signature-based detection and dynamic behavioral analysis. Its flexible, human-readable format allows it to be leveraged across various domains:

1. Incident Response (IR): During an active network breach, responders deploy YARA rules across the infrastructure to rapidly sweep systems for known threat indicators and map the extent of the compromise.
2. Threat Hunting: Security Operations Center (SOC) teams proactively deploy YARA rules across endpoints (using tools like Velociraptor or Wazuh) to hunt for dormant artifacts of Advanced Persistent Threats (APTs) or pre-ransomware staging behaviors.
3. Malware Research: Reverse engineers utilize YARA to classify newly acquired samples and correlate their code structures with known state-sponsored threat groups.
4. Digital Forensics: Forensic investigators utilize YARA to scan physical disk images or volatile memory dumps (RAM) to locate embedded payloads, hidden processes, or injected code that would not be visible on the file system.

Practical Application: Ransomware Detection To illustrate a real-world scenario, the following rule is designed to detect early indicators of a ransomware payload by correlating disparate strings:

None

```
rule Possible_Ransomware {  
    meta:  
        author = "SOC Analyst"  
        description = "Detects suspicious strings used in  
ransomware payloads"  
    strings:  
        $encrypt = "AES" nocase  
        $shadow = "vssadmin delete shadows" nocase
```

```
$ext = ".locked" nocase  
condition:  
    2 of ($encrypt, $shadow, $ext)  
}
```

This rule successfully identifies ransomware behavior by requiring at least two distinct indicators (e.g., the deletion of Windows shadow copies and the declaration of cryptographic routines) to be present, ensuring a high-fidelity alert.

2.2.6 The YARA Ecosystem and Supporting Tooling

The widespread adoption of YARA has fostered a rich ecosystem of open-source tools designed to generate, manage, and execute rules at scale:

- **yarGen**: Writing complex rules manually can be time-consuming. yarGen is an automated rule generator that extracts strings from a provided malware sample. Crucially, it incorporates a massive database of "goodware" strings (strings found in legitimate software). By removing these benign strings from the generated rule, yarGen significantly reduces the likelihood of false positives.
- **yarAnalyzer**: A companion to yarGen, yarAnalyzer automates the statistical analysis of YARA rule effectiveness, helping analysts evaluate signature overlaps and rule performance.
- **LOKI**: Developed by Florian Roth, LOKI is a free, open-source Indicator of Compromise (IoC) scanner. It performs endpoint scans utilizing four distinct methods: File Name Check, YARA Rule Check, Hash Check, and Command and Control (C2) Check, making it an invaluable tool for rapid triage.
- **THOR Lite**: A fast, multi-platform IoC and YARA scanner. While the enterprise version of THOR leverages the massive commercial Valhalla YARA rule base, the free THOR Lite version ships with a robust open-source signature base and includes modules for scanning file systems and process autoruns.
- **Rule Management (YAYA and YARA-CI)**: For organizations managing vast repositories of rules, YAYA is an open-source tool that allows researchers to import, merge, and manage

multiple YARA rulesets. Furthermore, YARA-CI integrates with GitHub to provide continuous integration testing for YARA rules, automatically identifying syntax errors and common false-positive triggers before deployment.

- **yextend**: Developed by Bayshore Networks, this extension expands YARA's capabilities to natively scan inside compressed archives (such as .zip or .tar), a common technique used by attackers to hide malicious payloads from network perimeter defenses.

Ultimately, YARA provides an unmatched level of scalability, reusability, and contextual power. By allowing security teams to define the very essence of a malware family, YARA has established itself as an indispensable asset in the defense of both traditional IT networks and critical OT infrastructures.

2.3 The Open Source Ecosystem for Malware Analysis

The detection methodologies discussed previously, such as Intrusion Detection Systems (IDS), are primarily designed to alert security teams to the presence of anomalous network traffic. However, to truly understand a cyber threat, especially an Advanced Persistent Threat (APT) targeting an Industrial Control System (IACS), an alert alone is insufficient. Security analysts must acquire the actual malicious payload, reverse-engineer its behavior, and extract highly specific Indicators of Compromise (IoCs) to generate custom YARA rules.

This analytical pipeline relies heavily on a global ecosystem of Open Source Intelligence (OSINT), collaborative malware repositories, and Threat Intelligence platforms. This subchapter explores the foundational pillars of this ecosystem—MalwareBazaar, VirusTotal, and vendor-driven Threat Intelligence (such as Kaspersky Lab)—and illustrates how they interact to facilitate advanced malware analysis in OT environments.

2.3.1 The Challenge of Payload Acquisition

For IT-security researchers and SOC analysts, OSINT is an invaluable resource. Cybersecurity blogs, whitepapers, and social media platforms frequently publish detailed analyses of emerging threats. However, researchers are often confronted with a severe logistical problem: the malware samples referenced in these reports are rarely easily accessible.

Historically, obtaining a specific malware sample required researchers to register across dozens of disparate antivirus scanning engines, sandboxes, and proprietary malware databases. Furthermore, many of these platforms impose severe download restrictions (limiting the number of samples a user

can retrieve per day) or lock their repositories behind expensive commercial paywalls. This paradigm created a significant bottleneck in threat research, particularly for independent analysts or academic researchers seeking to study targeted ICS malware. This fundamental industry challenge necessitated the creation of open, community-driven platforms.

2.3.2 MalwareBazaar: The Open Malware Repository

To address the limitations of commercial platforms, the cybersecurity community embraced MalwareBazaar, a platform launched in 2020 by Abuse.ch. Abuse.ch is a renowned Swiss non-profit cybersecurity project that operates several critical threat intelligence platforms, including ThreatFox (for IoC sharing), URLhaus (a malicious URL database), Feodo Tracker (a botnet Command and Control tracker), and the SSL Blacklist (SSLBL).

MalwareBazaar is designed with a singular, crucial mission: to collect known malicious samples, enrich them with additional intelligence, and provide them back to the community entirely for free. Unlike platforms that accept any file for scanning, MalwareBazaar exclusively hosts vetted malware samples. It explicitly rejects benign files and low-level annoyances like Adware or Potentially Unwanted Programs (PUP/PUA), ensuring the database remains a high-fidelity repository for serious threat research.

Operational Mechanics and Features:

- **Submission and Enrichment:** Any registered user can submit malware samples to the platform. Upon submission, the platform automatically extracts file metadata, calculates cryptographic hashes (MD5, SHA1, SHA256), and generates fuzzy hashes (such as TLSH and Imphash).
- **Querying Capabilities:** The database, which currently hosts over a million samples spanning hundreds of malware families, can be queried through various parameters. Analysts can search by specific hashes, file types (e.g., PE32 executables, Office documents, Python scripts), or malware family names (e.g., Emotet, LockBit). Crucially for OT security, analysts can utilize specific tags (such as #ICS, #SCADA, or #Modbus) to isolate threats designed to attack industrial environments.
- **Unrestricted Downloading:** Verified researchers can download an unlimited number of malware samples. To ensure safety and prevent accidental execution during the transfer process, all files are delivered in password-protected ZIP archives (standardized with the password "infected").
- **Automation via API:** For SOC environments and automated incident response workflows, MalwareBazaar provides a robust, free API. Security orchestration tools and SIEMs can

programmatically query the database for hashes or download daily malware batches using Python scripts or simple cURL commands.

2.3.3 VirusTotal: Multiscanning and Global Threat Intelligence

While MalwareBazaar excels at payload acquisition, VirusTotal remains the undisputed industry standard for initial triage and multiscanning. Created by the Spanish security company Hispasec Sistemas in June 2004, VirusTotal was recognized by PC World as one of the best 100 products of 2007. Its strategic importance in the cybersecurity landscape led to its acquisition by Google in 2012, and it currently operates under Google Security Operations.

VirusTotal operates on a multiscanning paradigm. It aggregates data from over 70 distinct antivirus products, heuristic engines, and URL/domain blocklisting services—referred to as Contributors. (Notably, in 2018, the Cyber National Mission Force, a unit subordinate to the U.S. Cyber Command, officially became a Contributor).

Core Functionalities and YARA Integration:

- **Unbiased Aggregation:** Any user can upload files (up to 650 MB) or submit suspicious URLs via a web interface, browser extensions, or a programmatic API. VirusTotal does not distribute or promote any specific third-party engine; it acts purely as an unbiased aggregator. This allows users to verify if their local antivirus software missed a threat or to identify false positives.
- **Dynamic Analysis:** Beyond static signature scanning, VirusTotal incorporates dynamic analysis capabilities utilizing tools like the Cuckoo Sandbox. This allows the platform to extract behavioral signals from the content, monitoring how a file behaves upon execution.
- **Information Sharing and Community:** Scanning reports are shared publicly, raising the global IT security baseline. A robust community network allows researchers to comment on files, share notes, and vote on whether particular content is harmful. Furthermore, antivirus vendors receive copies of files that bypassed their engines but were flagged by others, allowing them to update their signatures rapidly.
- **VirusTotal Intelligence (VTI) and YARA:** For premium users and enterprise customers, VirusTotal offers advanced criteria-based searching. Most importantly for this thesis, VirusTotal heavily integrates YARA. Through features known as LiveHunt and RetroHunt, malware analysts can deploy custom YARA rules against the massive VirusTotal corpus. This allows researchers to test the efficacy of a newly written YARA rule against millions of files, ensuring

the rule successfully catches the intended malware without generating unacceptable false positives.

The immense power of VirusTotal requires careful operational security (OpSec). Submitting a file to the public interface makes its contents available to premium partners. Mishandling this capability can lead to data exposure, as occurred in July 2023 when a staff member unintentionally exposed customer data. Organizations must ensure that sensitive OT configuration files or proprietary SCADA software are not uploaded to public scanners.

2.3.4 Threat Intelligence and OT-Specific Analysis (Kaspersky Lab & ICS CERT)

While platforms like VirusTotal and MalwareBazaar provide the raw data and the samples, the contextual intelligence required to understand what to look for is provided by specialized Threat Intelligence organizations.

In the realm of Industrial Control Systems, organizations such as Kaspersky Lab, Dragos, and various national ICS-CERTs (Computer Emergency Readiness Teams) play a foundational role. Kaspersky's ICS CERT, for instance, publishes exhaustive technical reports on the Tactics, Techniques, and Procedures (TTPs) utilized by APT groups targeting critical infrastructure.

These reports dissect complex, state-sponsored malware—such as Stuxnet, Industroyer/CrashOverride, and BlackEnergy—providing the community with actionable IoCs. They detail how these malicious agents exploit legacy industrial protocols, interact with specific Human-Machine Interfaces (HMIs), or attempt to rewrite PLC logic. By standardizing this intelligence using formats like STIX/TAXII and sharing it through platforms like MISP (Malware Information Sharing Platform), vendors enable independent researchers to track down affiliated malware samples in repositories like MalwareBazaar.

2.3.5 The Analytical Pipeline: From Sample to YARA Rule

The platforms described above do not operate in isolation; they form a cohesive, cyclical pipeline for malware analysis and defense generation. This workflow is the foundation of the experimental methodology utilized in this thesis:

1. Intelligence Gathering: An analyst reviews a threat intelligence report (e.g., from Kaspersky Lab) detailing a new APT campaign targeting OT networks, extracting the cryptographic hashes associated with the attack.

2. Triage and Verification: The analyst queries the hash on VirusTotal to observe which antivirus engines detect it, review community comments, and analyze its static properties and dynamic behavioral graphs.
3. Acquisition: Because downloading from VirusTotal is often restricted by commercial licenses, the analyst uses the hash to locate and freely download the raw, password-protected payload from MalwareBazaar.
4. Deep Analysis and Debugging: The downloaded sample is executed within an isolated, local sandbox environment (such as Cuckoo or CAPEv2) to safely monitor its runtime behavior, network callouts, and memory allocation.
5. Detection Engineering: Based on the unique strings, byte sequences, and behaviors extracted during the analysis, the analyst writes a highly specific YARA rule.
6. Validation: The YARA rule is tested locally and potentially against VirusTotal's RetroHunt to validate its accuracy.

Once this pipeline is complete, the final, critical challenge remains: how to deploy this newly created YARA rule across a live, distributed SCADA infrastructure to actively hunt for the threat. This operational deployment is achieved through advanced Digital Forensics and Incident Response (DFIR) frameworks, which will be explored in detail in the following subchapter through the introduction of the Velociraptor platform.

2.4 DFIR Tools: Introduction to Velociraptor

As cyber threats against Industrial Control Systems (ICS) evolve from generic malware to highly targeted Advanced Persistent Threats (APTs), the security paradigm must shift from passive monitoring to active threat hunting. Once a security analyst has acquired threat intelligence and generated a robust detection mechanism—such as the YARA rules discussed in previous subchapters—the immediate operational challenge becomes deployment. How can an analyst efficiently scan thousands of distributed, highly sensitive endpoints (such as SCADA servers, HMIs, and engineering workstations) for these specific Indicators of Compromise (IoCs) without disrupting critical industrial processes?

To address this challenge, modern security architectures employ Digital Forensics and Incident Response (DFIR) frameworks. Among the most advanced and highly regarded open-source platforms in this domain is Velociraptor.

2.4.1 What is Velociraptor?

Velociraptor is an advanced, open-source endpoint visibility and DFIR platform designed to provide deep, real-time insights into the state of a network's endpoints. Unlike traditional forensic tools that require physical access to a machine or the cumbersome extraction of full disk images, Velociraptor operates on a fundamentally different philosophy: it treats the endpoints themselves as the ultimate source of truth.

Developed to handle enterprise-scale deployments, Velociraptor allows incident responders to pose highly targeted queries to endpoints, aiming to directly progress an investigation and return only high-value data. This approach drastically reduces the network bandwidth required for data collection and minimizes the post-processing burden on centralized servers.

2.4.2 Architectural Deployment Models

Velociraptor is engineered for extreme flexibility and can be deployed in several configurations depending on the operational requirements and the strictness of the network segmentation (such as those dictated by the Purdue Model in OT environments).

Client-Server Deployment (Live Hunting): This is the most common and powerful deployment model. The Velociraptor agent (referred to as the client) is deployed as a lightweight background service on all target endpoints. The central Velociraptor server can be hosted on-premises (highly recommended for isolated OT networks) or in the cloud. The clients maintain a persistent, secure, and authenticated connection (typically via gRPC over mTLS) back to the server. This persistent connection allows investigators to rapidly iterate through the investigative process: tasking clients, collecting evidence, analyzing results, and re-collecting data in near real-time.

Offline Collector: In many critical OT environments, installing a persistent agent on a legacy HMI or a proprietary SCADA server is strictly prohibited due to stability concerns or vendor warranty invalidation. To solve this, Velociraptor allows the creation of Offline Collectors. These are preconfigured, standalone executable binaries designed to collect specific forensic artifacts without requiring installation. The collector is executed locally on the target machine, gathers the requested data and outputs it into a locally encrypted ZIP file. This file can then be manually exfiltrated and imported into the Velociraptor server for analysis.

Interactive Analysis (Dead Disk Forensics): Velociraptor is not limited to live systems. It can be used for the analysis of raw dead disk images (e.g., forensic copies of a compromised engineering

workstation). Using built-in remapping rules, a virtual client is launched which mounts and uses the disk image to collect the same artifacts as if the system were live. This ensures that forensic analysts can utilize identical procedures and queries regardless of whether they are analyzing a live network or a forensic backup.

2.4.3 The Core Differentiator: Velociraptor Query Language (VQL)

The true power, flexibility, and scalability of Velociraptor stem from the Velociraptor Query Language (VQL). VQL is a declarative query language, syntactically inspired by SQL, designed specifically for endpoint forensics.

The need for VQL arose from the limitations of legacy DFIR tools, which were often written in compiled languages or rigid scripting frameworks that were difficult to update rapidly during an active incident. VQL allows users to seamlessly interrogate the underlying operating system. Through VQL, an analyst can query active processes, parse the NTFS Master File Table (MFT), read the Windows Registry, extract Windows Event Logs, and dump process memory.

Instead of writing a custom Python script, deploying it, managing its dependencies, and parsing its unstructured output, an analyst can write a simple VQL query. Because VQL is evaluated natively by the Velociraptor client, it inherently respects the platform's safety guardrails, including strict CPU limits and memory resource controls—features that are absolutely critical to ensure that a forensic scan does not inadvertently crash a fragile ICS endpoint.

2.4.4 Velociraptor Artifacts: Encapsulating Forensic Knowledge

While VQL is the engine, Artifacts are the steering wheel. At its core, an Artifact is a YAML file that encapsulates one or more VQL queries alongside human-readable metadata, parameters, and descriptions.

In a client-server deployment, analysts do not run raw VQL directly on the endpoints. Instead, they trigger Artifacts. The server compiles the Artifact into raw VQL and pushes it to the client for evaluation. This architecture provides profound benefits for incident response workflows:

- **Encapsulation and Reusability:** Once an Artifact is written to detect a specific threat (e.g., a specific Modbus manipulation script), the user does not need to remember the underlying VQL syntax. They simply call the Artifact by name. Artifacts function like modular Lego bricks that can be chained together.

- **Parameterization:** Artifacts can declare parameters (such as file paths, regex patterns, or specific IP addresses) allowing users to customize the query's behavior directly from the GUI prior to execution.
- **Operational Safety:** In OT environments, safety is paramount. Artifacts can define preconditions—lightweight VQL queries that run first to determine if the main query is relevant or safe to execute on a specific endpoint. For example, an Artifact designed to scan Windows Event Logs might have a precondition that ensures the target is actually running Windows; if the target is a Linux-based ICS firewall, the collection is safely skipped.
- **Community Collaboration:** Because Artifacts are simply YAML files, they are highly portable. Platforms like the Velociraptor Artifact Exchange allow DFIR professionals worldwide to share detection logic. When a new zero-day vulnerability is announced, the community often produces and shares a Velociraptor Artifact to detect it within hours.

2.4.5 The Velociraptor Ecosystem and Integration

Velociraptor is designed as an open system, serving as a central hub within a broader security architecture.

- **Integration with External Tools:** Artifacts can seamlessly wrap and execute external third-party tools. Velociraptor handles the logistics of securely pushing the binaries to the endpoint, executing them, and returning the structured results to the server. This allows teams to utilize specialized tools like Nextron Thor (a commercial IoC scanner) or Hayabusa (a fast Windows event log analyzer) securely and at scale.
- **Data Export and Automation:** The data collected by Velociraptor (which is already structured by VQL into rows and columns) can be automatically exported to data lakes, SIEMs (like Splunk or Elastic), or collaborative analysis platforms like Timesketch.
- **API Control:** Velociraptor can be fully automated using a gRPC-based API, allowing it to be controlled via Python or Go. This enables seamless integration into Security Orchestration, Automation, and Response (SOAR) workflows, allowing a network alert (e.g., from an IDS) to automatically trigger a forensic collection via Velociraptor without human intervention.

2.4.6 YARA Integration in Velociraptor

The intersection of YARA (discussed in Subchapter 2.2) and Velociraptor represents one of the most potent capabilities available to modern threat hunters. Velociraptor natively embeds the YARA engine, allowing analysts to execute complex pattern-matching rules across the entire network fleet simultaneously.

Velociraptor exposes YARA scanning through VQL plugins, primarily `yara()` for scanning files on disk and `proc_yara()` for scanning the volatile memory of running processes.

File System Scanning with YARA The `yara()` VQL plugin allows analysts to scan the file system. When scanning standard files on disk, Velociraptor utilizes memory mapping (`mmap()`), allowing the YARA engine to optimize the scan across the entire file for maximum performance.

However, Velociraptor's true flexibility shines through its use of Accessors. Accessors are abstraction layers that tell VQL how to read data. By passing an accessor to the `yara()` plugin, an analyst can apply a YARA rule not just to standard files, but to unstructured data streams. For example, YARA rules can be applied to the raw NTFS volume (using the `ntfs` accessor) to find malware hiding in unallocated space, or directly to Windows Registry values (using the `registry` accessor) to detect fileless malware persistence mechanisms.

Example Scenario: An analyst suspects an ICS engineering workstation was compromised via a malicious URL. To identify the threat, the analyst deploys a VQL query that uses a `glob` to find all Edge browser cache files, and then pipes those files into the `yara()` plugin utilizing a custom YARA rule designed to extract obfuscated URLs.

Memory Scanning with YARA Modern APTs frequently employ "fileless" techniques, where the malicious payload is never written to the physical hard drive, but instead is injected directly into the memory (RAM) of a legitimate process (such as `explorer.exe` or a SCADA interface software). Traditional antivirus scans of the hard drive will completely miss these threats.

By utilizing the `proc_yara()` plugin, Velociraptor can iterate through all running processes on an endpoint and apply YARA rules directly to their allocated virtual memory space. This technique is critical for identifying sophisticated banking trojans, ransomware, and ICS-specific espionage tools operating stealthily in memory.

2.4.7 Best Practices for YARA Hunting in OT Environments

While Velociraptor makes deploying YARA rules across thousands of endpoints trivial, doing so in an Operational Technology environment requires strict adherence to best practices to ensure stability:

1. **Rule Tuning and False Positive Reduction:** YARA rules must be highly specific. A loose rule deployed across a fleet will generate massive amounts of false positives, rendering the results useless and generating unnecessary network traffic. Rules should always be tested locally or against platforms like VirusTotal before enterprise deployment.
2. **Targeted Scanning via VQL:** YARA scanning is computationally expensive because it requires reading significant amounts of data. Instead of blindly scanning the entire C:\ drive, analysts must leverage VQL to target the scan. By combining VQL logic with YARA, an analyst can instruct Velociraptor to only scan files created in the last 7 days, or only files located in the \AppData\ directories of active users, drastically reducing disk I/O.
3. **Client-Side Throttling:** In SCADA environments, endpoint CPU cycles are critical for maintaining real-time control over physical processes. Velociraptor provides built-in Ops/Sec (Operations Security) mechanisms to enforce strict client-side throttling. A hunt can be configured to pause if CPU utilization exceeds a certain threshold, ensuring that the forensic scan never impacts the availability or integrity of the industrial system.

By combining the highly descriptive pattern-matching capabilities of YARA with the surgical, distributed querying power of Velociraptor, security teams can proactively hunt for complex threats within the OT environment, validating alerts generated by perimeter defenses and ensuring the continuous operational resilience of critical infrastructure.

3: Design and Setup of the Experimental Environment

To effectively test and validate advanced detection mechanisms like YARA and Velociraptor within an Operational Technology context, a highly realistic testing ground is required. Testing offensive techniques and deploying experimental Endpoint Detection and Response (EDR) solutions directly on a live production plant is practically impossible due to the critical requirement for continuous availability and the severe physical risks involved. To overcome this limitation, the experimental activities of this thesis were conducted within the Relatech Lab-OT Excellence Center.

This laboratory is engineered not merely as a collection of isolated virtual machines, but as a comprehensive "Digital Twin" of a modern industrial enterprise, replicating both the IT and OT network infrastructures, the segmentation firewalls, and the industrial control processes.

3.1. The "Digital Twin" Concept: Virtualization and Storage Infrastructure

The foundation of the experimental environment is built upon a robust virtualization architecture powered by Proxmox Virtual Environment (VE), a Type-1 open-source hypervisor. Proxmox VE was selected for its flexibility in managing both full virtual machines (KVM) and lightweight containers (LXC), allowing for the simulation of a complex, multi-node enterprise network on a single physical host.

The physical host supporting the hypervisor is equipped with enterprise-grade hardware, specifically dual Intel Xeon processors, 128 GB of ECC RAM, and a high-speed NVMe storage array, providing the necessary computational power to run over a dozen simultaneous virtual machines and network appliances without experiencing performance bottlenecks.

A critical aspect of this Digital Twin is its advanced storage architecture, which enables aggressive penetration testing and rapid state recovery. The hypervisor is integrated with a QNAP Network Attached Storage (NAS) via the iSCSI protocol. The storage is configured using LVM-Thin (Thin Provisioning). Unlike static storage allocation, thin provisioning utilizes a "Copy-on-Write" mechanism.

This storage configuration is paramount for cybersecurity research. It allows the execution of instantaneous "Snapshots" of the entire infrastructure. During the experimental phases—particularly when deploying malware, executing lateral movement, or testing the False Data Injection (FDI) attack—the system state is frequently compromised. The LVM-Thin snapshot capability allows the researcher to safely detonate malicious payloads and, once the forensic data is collected by the EDR, perform an immediate rollback to a pristine, pre-attack state, ensuring the reproducibility of the experiments.

3.2. Network Architecture and IT/OT Segregation

The network topology of the laboratory is meticulously designed to emulate the strict hierarchical zones defined by the Purdue Enterprise Reference Architecture (PERA). To achieve this, the hypervisor utilizes Open vSwitch (OVS) bridges to create isolated Virtual Local Area Networks (VLANs), which are interconnected and regulated by multiple virtualized enterprise firewalls from different vendors, including FortiGate, PfSense, and Stormshield.

The infrastructure is broadly divided into two macro-environments:

1. **The IT Network:** This segment simulates the corporate environment and is divided into three primary VLANs:
 - *VLAN 201 (Client IT):* Administrative desktops and employee workstations.
 - *VLAN 202 (Server IT):* Internal corporate servers, including the SMTP server, monitoring tools (Centreon), and the core Active Directory Domain Controller.
 - *VLAN 203 (DMZ):* The Demilitarized Zone housing externally facing web servers and services.
2. **The OT Network:** This segment simulates the industrial plant floor and its supervisory systems:
 - *VLAN 204 (OT Server):* Industrial servers, log aggregators (Graylog), and the NTP Server.
 - *VLAN 205 (OT Remote):* Dedicated to secure remote access for OT operators.
 - *VLAN 206 (OT Network):* The deepest level of the architecture (Purdue Levels 1 and 2), housing the PLCs and the SCADA Human-Machine Interface (HMI).

While a perfectly configured perimeter would block most lateral movement attempts, real-world ICS incidents often exploit permissive firewall rules left open for maintenance or legacy IT-OT integration. To construct a realistic attack scenario for this thesis, the perimeter firewalls were intentionally configured with a flawed, permissive baseline.

Specifically, while general internet traffic is blocked from reaching the OT network, specific routing rules allow lateral movement:

- A rule allows SSH and RDP traffic from the compromised Web Server in the DMZ (VLAN 203) to the IT Server subnet (VLAN 202), mimicking an oversight in DMZ isolation.
- A subsequent rule permits authentication traffic and Modbus/TCP traffic from the IT Server subnet (VLAN 202) to cross the IT/OT boundary firewall and reach the HMI and PLCs in the OT Network (VLAN 206). This emulates a common architectural flaw where corporate domains are excessively trusted by the industrial control layer, setting the stage for the attack path described in Chapter 4.

3.3. Emulating the Industrial Process: Active Directory, PLCs, and SCADA HMI

The endpoints within the virtualized VLANs were configured to replicate the software stack of an operational industrial facility.

The IT Layer: Active Directory At the heart of the IT network (VLAN 202) resides a Windows Server 2016 machine configured as the Primary Domain Controller (DC) for the Active Directory (AD) forest. In modern converged environments, AD is frequently used to manage user authentication for both IT workstations and OT supervisory software. The inclusion of this component is critical for the experimental attack chain, as it provides a high-value target for privilege escalation (via the Zerologon vulnerability) and Kerberos ticket manipulation.

The OT Layer: PLCs and SCADA HMI The industrial control logic is executed by software-based Programmable Logic Controllers (PLCs). The laboratory employs Ubuntu virtual machines running OpenPLC, an open-source industrial controller that natively supports standard industrial protocols. The setup includes nodes emulating a Siemens S7-300 architecture and a standard Modbus TCP controller. To supervise the physical process, the OT Network (VLAN 206) hosts a SCADA Human-Machine Interface (HMI). For this research, ScadaBR—an open-source SCADA application running on an Apache Tomcat Java Virtual Machine (JVM)—was deployed. ScadaBR actively polls the PLCs to update its graphical interface and log historical data.

The Simulated Physical Process: Beer Bottling Plant To provide semantic context for the False Data Injection (FDI) attack, the OpenPLC logic was programmed to simulate a continuous manufacturing process: an automated beer bottling and capping line. The SCADA system monitors and controls the Modbus TCP registers associated with this physical process. The Modbus map (Table 3.1) dictates how the physical variables are translated into memory addresses.

Table 3.1: Modbus Memory Map for the Simulated Bottling Process

Data Point Name	Modbus Register Type	Register ID	Description
Sensore_temperature	Input Register (16-bit)	13	Monitors the beer pasteurization temperature (°C).
Sensore_pressure	Input Register (16-bit)	14	Monitors the pressure of the CO2 carbonation tank (Bar).
Sensore_level	Input Register (16-bit)	15	Monitors the fluid level in the primary filling vat (%).
pumpCmd	Coil (Binary)	16	Command register to start/stop the main filling pump.

pumpStatus	Discrete Input (Binary)	17	Feedback sensor indicating if the pump is actively running.
-------------------	----------------------------	----	---

This contextualization is vital. When the FDI attack is executed in the following chapters, manipulating register ID 13 will logically simulate a critical temperature spike in the pasteurization process, providing a tangible metric to evaluate the effectiveness of the proposed EDR detection mechanism.

3.4. Attack and Defense Tooling Setup (Kali Linux and Velociraptor EDR)

To simulate a realistic Advanced Persistent Threat (APT) lifecycle and evaluate the proposed detection mechanisms, the experimental environment was provisioned with dedicated offensive and defensive toolsets. This setup allowed for a continuous "Red Team vs. Blue Team" dynamic within the Digital Twin.

Offensive Setup: The Red Team Infrastructure The offensive operations were conducted from a dedicated penetration testing virtual machine running Kali Linux. This machine was strategically positioned within the IT administrative subnet to simulate an internal threat or a compromised corporate workstation initiating lateral movement. The Kali Linux instance was equipped with standard offensive frameworks, notably Metasploit for exploit delivery and payload generation, and Impacket for Active Directory enumeration and Kerberos ticket manipulation (Pass-the-Ticket attacks). To navigate the segmented network architecture and reach the isolated OT VLANs, tools like Proxychains and dynamic SSH port forwarding were heavily utilized to create encrypted tunnels through compromised intermediary nodes.

Defensive Setup: Velociraptor EDR Deployment The core of the Blue Team infrastructure—and the primary engine for the Semantic Memory Hunting proposed in this thesis—was the Velociraptor Endpoint Detection and Response (EDR) platform. The deployment of Velociraptor was architected to mirror an enterprise-grade Security Operations Center (SOC) configuration:

- **The Velociraptor Server:** The central management server was deployed on a dedicated Linux virtual machine located in the IT Server subnet (VLAN 202). This positioning is crucial: it ensures the management console is securely segregated from the highly vulnerable DMZ and the operationally sensitive OT network, while remaining accessible to IT security administrators.
- **The Velociraptor Agents (Clients):** The lightweight Velociraptor client service was deployed on the most critical endpoints across the infrastructure boundaries. Specifically, agents were

installed on the compromised Web Server residing in the DMZ (VLAN 203), and crucially, on the Linux machine hosting the SCADA HMI (ScadaBR) in the OT Network (VLAN 206).

The agents were configured to maintain a persistent, secure gRPC connection back to the server in VLAN 202. This architecture allowed the centralized server to continuously task the HMI endpoint with custom VQL (Velociraptor Query Language) artifacts. By pushing the analytic workload directly to the endpoint agent, the architecture minimizes network congestion—a critical requirement in ICS environments—while enabling real-time, deep visibility into the memory space of the SCADA application.

3.5. Vulnerability Analysis of the Modbus TCP Protocol

To fully understand the mechanics of the False Data Injection (FDI) attack executed in this research, it is essential to analyze the structural vulnerabilities of the industrial protocol targeted: Modbus TCP.

Originally developed in 1979 by Modicon (now Schneider Electric) for serial communication between Programmable Logic Controllers (PLCs), Modbus has become the de facto standard for industrial automation due to its simplicity and open nature. When the industry transitioned to Ethernet-based networks, the protocol was adapted into Modbus TCP (operating on port 502) simply by encapsulating the original serial message within a TCP/IP wrapper. However, this adaptation carried over the protocol's original design philosophy, which assumed that the physical network was inherently secure and isolated. Consequently, Modbus TCP was designed entirely without built-in security mechanisms, making it fundamentally inadequate for modern, converged IT/OT environments.

3.5.1 Structural Anatomy of a Modbus TCP Packet

A standard Modbus TCP packet consists of two main sections: the MBAP Header and the Protocol Data Unit (PDU).

1. MBAP Header (Modbus Application Protocol): This 7-byte header is specifically added to allow Modbus to traverse TCP/IP networks. It comprises four critical fields:

- *Transaction Identifier (2 bytes)*: Used to pair requests and responses. If the SCADA Master sends a request with ID 0x1A2B, the PLC Slave must reply with the exact same ID. If this identifier is altered during an interception, the HMI will discard the response.
- *Protocol Identifier (2 bytes)*: Identifies the protocol type. For Modbus, this value is strictly 0x0000. In a network forensics context, this static value serves as an excellent signature for identifying Modbus traffic within a generic packet capture.

- *Length (2 bytes)*: Indicates the number of bytes remaining in the packet, necessary for the receiver to determine the end of the message.
- *Unit Identifier (1 byte)*: Historically used in serial networks to identify specific slave devices on a daisy-chain. In TCP networks, the IP address handles routing, but this byte is retained for legacy compatibility or to address serial devices behind a TCP gateway.

2. PDU (Protocol Data Unit): The PDU contains the core industrial instruction and is identical to the payload used in serial Modbus. It consists of:

- *Function Code (1 byte)*: The specific command instructing the PLC on what action to perform. Common function codes targeted by attackers include:
 - 0x01 (Read Coils) and 0x02 (Read Discrete Inputs): To monitor binary states.
 - 0x03 (Read Holding Registers): To read read/write 16-bit configuration parameters.
 - 0x04 (Read Input Registers): To read read-only 16-bit physical sensor data (e.g., the temperature sensor used in this lab).
 - 0x05 (Write Single Coil) and 0x06 (Write Single Register): Used to actively manipulate the physical process (e.g., turning off a pump or changing a setpoint).
- *Data (Variable Length)*: The payload content, which varies entirely based on the Function Code and whether the packet is a Request or a Response.

3.5.2 Inherited Security Vulnerabilities

Because Modbus TCP lacks "Security by Design," it exposes the industrial architecture to several critical threat vectors that are actively exploited in Advanced Persistent Threats (APTs) targeting SCADA systems:

- **Lack of Encryption (Cleartext Communication)**: As highlighted in the thesis overview, Modbus TCP transmits all data, including critical sensor readings and command payloads, in plain text. This makes the protocol highly susceptible to Sniffing. An attacker positioned on the OT network can passively monitor the traffic (e.g., using Wireshark) and map the entire industrial process without generating any anomalous network noise.
- **Lack of Authentication**: The protocol relies entirely on IP addressing for trust. There is no cryptographic handshake, username/password exchange, or Certificate Authority involved. Any device capable of routing a TCP packet to port 502 of the PLC can issue valid Write commands (e.g., Function Code 0x06). If an attacker compromises an endpoint within the OT network—or tunnels traffic through the DMZ, as demonstrated in this research—they can manipulate the physical process effortlessly.

- **Lack of Integrity Checks:** Modbus TCP does not employ digital signatures or Message Authentication Codes (MAC). The receiver has no mechanism to verify if the packet was altered in transit. This specific flaw makes the protocol exceptionally vulnerable to Man-in-the-Middle (MitM) and False Data Injection (FDI) attacks.

As will be detailed in Chapter 4, this lack of integrity verification allows an attacker to intercept the legitimate Response packet sent by the PLC, modify the Data field, and forward it to the SCADA HMI. Because the Transaction ID and the IP headers remain valid, the HMI accepts the fraudulent data as absolute truth, effectively "blinding" the human operator to the real physical state of the plant.

4. Advanced Threat Modeling: The "Stuxnet-like" Attack Scenario

To rigorously evaluate the efficacy of both network-based and endpoint-based detection mechanisms in an Operational Technology environment, a sophisticated, multi-stage attack scenario was modeled and executed within the Digital Twin laboratory. This scenario mirrors the Tactics, Techniques, and Procedures (TTPs) of modern Advanced Persistent Threats (APTs) targeting critical infrastructure.

The primary objective of the attack was not merely to disrupt the network, but to execute a "Stuxnet-like" False Data Injection (FDI). The ultimate goal was to silently manipulate the physical state of the PLCs controlling the simulated industrial process, while simultaneously spoofing the SCADA Human-Machine Interface (HMI) to display normal operating parameters to the human operator.

4.1. Initial Compromise and Lateral Movement (From DMZ to Active Directory)

The attack chain was initiated under the assumption that a perimeter breach had already occurred, aligning with the modern "Assume Breach" cybersecurity paradigm. In real-world industrial scenarios, internet-facing assets are frequently the first point of compromise. Accordingly, the initial foothold in this simulation was established on a vulnerable Web Server located within the corporate IT Demilitarized Zone (DMZ, VLAN 203).

Initial Access and Reverse Shell Deployment Following an initial reconnaissance phase that mapped the external perimeter, a Remote Code Execution (RCE) vulnerability was identified on the DMZ Web Server. To exploit this flaw and gain interactive control over the machine, a custom malicious payload was generated using the Metasploit Framework's msfvenom utility. A 64-bit Linux Executable and Linkable Format (ELF) payload was crafted to initiate a reverse TCP connection (linux/x64/meterpreter/reverse_tcp) back to the attacker's Kali Linux machine.

To deliver the payload (`rev_shell_64.elf`), a temporary HTTP server was hosted on the attacker's machine using Python. Leveraging the RCE vulnerability on the target web server, a concatenated shell command was executed to download the payload via `wget`, grant it execution permissions (`chmod +x`), and execute it from the `/tmp/` directory. Upon execution, the payload successfully called back to the Metasploit handler, establishing a covert, encrypted meterpreter session with elevated privileges on the DMZ server.

Pivoting and Network Traversal At this stage, the attacker resided on the DMZ Web Server. However, the critical OT assets (PLCs and HMI) and the core IT management servers (such as the Active Directory Domain Controller) were strictly segregated in different VLANs and could not be reached directly from the attacker's external IP address. To traverse the network boundaries and bypass the perimeter firewalls, the attacker utilized the compromised web server as a pivot point.

Pivoting is a fundamental lateral movement technique that routes traffic from the attacker's machine through a compromised host, effectively using it as a proxy to access otherwise unreachable internal subnets. Within the Metasploit console, a dynamic route was injected (`route add 192.168.202.0 255.255.255.0 1`) to explicitly tunnel all traffic destined for the IT Server subnet (VLAN 202) through the active meterpreter session.

To enable external post-exploitation tools—such as the Impacket suite required for Active Directory exploitation—to utilize this route, an auxiliary SOCKS5 proxy server was launched within Metasploit (`auxiliary/server/socks_proxy`), bound to the local port 1080 on the Kali machine. By configuring the `proxychains` utility to route outbound traffic through this local SOCKS5 proxy, the attacker successfully created a stable, encrypted tunnel. This allowed all subsequent offensive commands to transparently traverse the DMZ pivot, effectively bridging the gap between the external attack machine and the internal Active Directory Domain Controller, setting the stage for the privilege escalation phase.

4.2. Privilege Escalation via Domain Controller Compromise

To successfully breach the isolated Operational Technology (OT) network, the attacker required highly privileged credentials. In the converged IT/OT architecture of the Digital Twin, the Windows Server 2016 Active Directory (AD) Domain Controller (DC) located in the IT Server subnet manages authentication for both IT administrators and OT supervisory accounts. Consequently, compromising the DC grants absolute control over the entire infrastructure, enabling seamless lateral movement.

4.2.1 Exploiting CVE-2020-1472 (Zerologon)

The attacker targeted the Domain Controller (LabOT-DC, IP: 192.168.202.3) by exploiting Zerologon (CVE-2020-1472), a critical vulnerability within the Netlogon Remote Protocol (MS-NRPC). This vulnerability stems from a flawed cryptographic implementation of the AES-CFB8 algorithm, which utilizes a predictable initialization vector (all zeros). By sending a continuous stream of zeros, an attacker can reliably bypass authentication and spoof the identity of any domain-joined computer, including the DC itself.

Routing the attack through the SOCKS5 proxy established in the DMZ, the attacker utilized a Metasploit auxiliary module (`auxiliary/admin/dcerpc/cve_2020_1472_zerologon`). The exploit successfully bypassed the Netlogon authentication mechanism and executed a password change request, forcibly resetting the password of the Domain Controller's own machine account (LabOT-DC\$) to an empty string within the Active Directory database.

4.2.2 NTDS Dumping and Operational Security (OpSec)

With the DC machine account password effectively nullified, the attacker proceeded to extract the domain's credential database. Utilizing the `impacket-secretsdump` script via `proxychains`, the attacker authenticated to the DC using the empty password and initiated a clone of the NTDS.dit database using the DRSUAPI protocol.

This operation successfully dumped the NTLM hashes and Kerberos encryption keys for all domain users. From this output, the attacker acquired the NTLM hash of the absolute domain Administrator and, crucially, the `admin_hmi` account—a highly privileged OT service account necessary to access the SCADA Human-Machine Interface.

Crucial Operational Security (OpSec) Procedure: Executing a Zerologon attack creates a critical desynchronization: while the AD database (NTDS.dit) registers an empty password for the DC, the Windows operating system retains its original, complex password within the local Local Security Authority (LSA) registry. Upon reboot or ticket renewal, this fracture would cause a catastrophic Denial of Service (DoS) across the domain. To maintain stealth and preserve the operational availability of the industrial plant, the attacker performed a mandatory cleanup procedure. Leveraging the newly acquired Administrator NTLM hash, a Pass-the-Hash attack was executed to dump the local LSA secrets of the DC. The original plaintext machine password (`$MACHINE.ACC`) was extracted in hexadecimal format and seamlessly re-injected into the Active Directory database using a custom Python restoration script (`restorepassword.py`). This sophisticated procedure closed the vulnerability, leaving the Domain Controller fully functional but entirely compromised.

4.2.3 Overpass-the-Hash (Kerberos Ticket Generation)

Equipped with the NTLM hash of the admin_hmi account, the attacker prepared to access the Linux-based SCADA HMI. However, unlike Windows SMB or WMI services, standard SSH daemons do not natively support NTLM Pass-the-Hash authentication. To bypass this limitation, an advanced technique known as Overpass-the-Hash (or Pass-the-Ticket) was employed.

Because the Linux HMI was integrated into the Active Directory domain via SSSD (System Security Services Daemon), it accepted Kerberos authentication. The attacker utilized the `impacket-getTGT` utility to request a Kerberos Ticket Granting Ticket (TGT) directly from the Domain Controller on behalf of admin_hmi, utilizing the stolen NTLM hash as proof of identity. Subsequently, `impacket-getST` was used to request a Service Ticket (TGS) specifically scoped for the SSH service of the HMI target (SPN: host/hmi).

The generated Kerberos credential cache (.ccache) was exported into the attacker's local environment variables. Finally, the attacker initiated an SSH connection utilizing the GSSAPI authentication flag (`ssh -K`). To ensure the connection successfully traversed the segmented network without triggering DNS resolution errors through the tunnel, the proxy routing was handled natively by appending an `ncat` proxy command (`-o ProxyCommand="ncat --proxy-type socks5..."`).

This complex cryptographic maneuver allowed the attacker to obtain an interactive root-level shell on the highly isolated SCADA HMI, entirely bypassing password requirements and bridging the gap between the IT and OT networks.

4.3. The False Data Injection (FDI) Attack: Modbus Manipulation via NetfilterQueue (MITM)

Having successfully obtained a persistent, privileged root shell on the SCADA Human-Machine Interface (HMI), the attacker moved to the core objective of the operation: manipulating the physical state of the industrial process while blinding the supervisory system.

Because the Modbus TCP protocol transmits data in cleartext and completely lacks integrity verification mechanisms, it is highly susceptible to False Data Injection (FDI). However, executing a Man-in-the-Middle (MitM) attack via traditional network spoofing (e.g., ARP poisoning) can be noisy and easily detected by modern perimeter defenses or network switches. To maximize stealth, the attacker opted for a localized, host-based MitM attack directly within the network stack of the compromised Linux HMI.

4.3.1 Sideloaded Dependencies in an Air-Gapped Environment

Executing advanced packet manipulation requires specific libraries, namely `scapy` for packet crafting and `NetfilterQueue` for interacting with the Linux kernel's firewall queues. In a properly segmented Operational Technology environment, endpoints like the HMI are denied outbound internet access. Attempting to download these dependencies via standard package managers (e.g., `apt` or `pip`) would fail or trigger perimeter alarms.

To maintain stealth, the attacker utilized a local Docker container running an identical Ubuntu distribution to resolve and download the exact dependency tree (Debian `.deb` packages and Python `.whl` files) on their own machine. These offline packages were then pushed to the HMI's `/tmp` directory via SCP, leveraging the Kerberized SSH tunnel (`scp -o GSSAPIAuthentication=yes`). The dependencies were installed locally using `dpkg`, completely bypassing the need for external network connectivity.

4.3.2 Host-Based Traffic Interception (IPTables)

The ScadaBR application continuously polls the Modbus TCP PLCs to update its graphical interface and historical logs. To intercept this telemetry before it reaches the Java Virtual Machine (JVM) hosting ScadaBR, the attacker manipulated the local Linux firewall by injecting a routing rule:

```
sudo iptables -I INPUT 1 -p tcp --sport 502 -j NFQUEUE --queue-num 1
```

This iptables rule instructs the Linux kernel to capture all incoming TCP traffic originating from port 502 (Modbus responses from the PLCs) and divert it into a dedicated Netfilter queue (queue 1), effectively pausing the packet's journey to the application layer.

4.3.3 Packet Mangling and False Data Injection

With the traffic trapped in the kernel queue, the attacker executed a custom Python script designed to bind to the Netfilter queue, parse the industrial data, and perform surgical modifications.

1. **Packet Parsing:** Utilizing the `scapy` library, the script reassembled the IP and TCP layers to access the Modbus TCP payload. It parsed the 7-byte MBAP Header to extract the Transaction Identifier and length, ensuring synchronization.
2. **Target Identification:** The script analyzed the Protocol Data Unit (PDU) to identify specific Function Codes (e.g., 0x04 Read Input Registers) containing the physical process variables. By mapping the data array, the script pinpointed the exact byte sequence representing critical telemetry, such as the pasteurization temperature (Register ID 13).

3. Data Overwrite: The legitimate physical value reported by the PLC (e.g., 50.0°C) was discarded and overwritten with a fraudulent, hardcoded payload (e.g., 777.0°C).
4. Checksum Recalculation and Release: Because the payload length or data content was altered, the script recalculated the TCP checksums to prevent the packet from being dropped as corrupted by the local network stack. Finally, the script executed `packet.accept()`.

The altered packet was released from the Netfilter queue and delivered to the ScadaBR application. Because the IP headers and the Modbus Transaction IDs remained perfectly valid and synchronized, the SCADA software accepted the forged packet as absolute truth. Through this host-based FDI attack, the adversary successfully hijacked the telemetry loop, asserting total control over what the human operator perceives on the HMI dashboard.

4.4. HMI Spoofing: Deceiving the SCADA Operator

The culmination of the attack chain resulted in a complete disconnect between the physical reality of the industrial process and the supervisory dashboard. By successfully hijacking the telemetry loop via the localized Man-in-the-Middle attack, the adversary achieved what is technically defined as "UI Spoofing" or "Sensory Blindness," a highly sophisticated technique notoriously pioneered by the Stuxnet malware.

To cause physical disruption, an APT must execute two simultaneous actions: malicious actuation and operator deception. First, having established a foothold on the OT network, the attacker can leverage standard Python scripts (such as `pymodbus` or specific ICS attack frameworks) to send unauthorized Modbus write commands directly to the PLC. For example, the attacker can send a Function Code 0x05 (Write Single Coil) to arbitrarily halt the main cooling pump (Register ID 16), or a Function Code 0x06 (Write Single Register) to dangerously alter a pressure setpoint.

In a normal scenario, the PLC would immediately reflect this critical state change. The SCADA software, during its polling cycle, would read the anomalous pressure or temperature, trigger programmed thresholds, log the event, and flash visual alarms on the HMI dashboard to alert the human operator.

Neutralizing the Human Element However, due to the NetfilterQueue packet mangling script running stealthily in the background of the HMI operating system, this defensive feedback loop is entirely neutralized.

When the PLC attempts to report the critical temperature spike resulting from the halted cooling pump, the script intercepts the Modbus response packet before it reaches the ScadaBR Java application. The

script identifies the specific byte sequence corresponding to the temperature register and overwrites the real, dangerous value with a hardcoded, "safe" historical value (e.g., a steady 50.0°C).

The forged packet is then passed to the SCADA software. Consequently, the HMI processes the data normally, no digital alarms are triggered, and the dashboard remains entirely "green." The human operator, completely reliant on the HMI interface, continues to perceive normal operating conditions while the physical plant approaches catastrophic failure.

This specific attack scenario perfectly illustrates the dire limitations of standard perimeter security and traditional monitoring. Because the manipulative code (the Python script) was executed locally within the HMI's network stack, and the malicious commands utilized legitimate, unauthenticated Modbus protocols, traditional Network Intrusion Detection Systems (NIDS) were entirely blind to the deception. Furthermore, standard IT endpoint protections (such as traditional antivirus) would not flag a standard Python script modifying local network routes as definitive malware.

To detect this highly targeted, process-aware manipulation, a paradigm shift in defensive strategies is required. Security teams must transcend static file scanning and network signature matching. As detailed in the following chapter, the solution lies in reprogramming Endpoint Detection and Response (EDR) platforms like Velociraptor to become "Physics-Aware"—bypassing the spoofed network stack entirely to semantically validate the industrial data directly within the volatile memory (RAM) of the SCADA application

5. Detection Strategies: From Known Malware to Semantic Memory Hunting

The offensive simulation detailed in Chapter 4 demonstrated how an adversary could breach the IT/OT boundary and establish a covert foothold on a critical industrial asset. To counter such threats, a robust defensive architecture must be deployed. This chapter outlines the evolution of the detection strategies implemented within the experimental Digital Twin. The methodology progresses from evaluating traditional, network-based YARA scanning against known ICS malware, identifying its fundamental structural limitations, and ultimately culminating in the design of a novel Endpoint Detection and Response (EDR) mechanism: Semantic Memory Hunting.

5.1 Evaluating Network-Based Detection on Known Malware

The initial phase of the defensive evaluation focused on traditional Network Intrusion Detection Systems (NIDS) operating at the IT/OT boundary. The theoretical objective was to intercept malicious payloads in transit before they could reach the vulnerable industrial endpoints. To execute a

comprehensive benchmark, a corpus of historically significant ICS malware samples including variants of Stuxnet, Havex, BlackEnergy, Industroyer, and EKANS was introduced into the test environment, alongside modern threats targeting the Modbus protocol.

5.1.1 Acquisition and Reverse Engineering of ICS Payloads.

A primary test case for this evaluation involved FrostyGoop, a recently discovered ICS malware written in Golang and specifically engineered to interact directly with Modbus TCP networks. The raw malicious payload was acquired from the open-source threat intelligence repository, MalwareBazaar.

The first step in detection engineering is the creation of a reliable signature. Initial attempts to generate automated YARA signatures using the yarGen utility yielded rules that were excessively generic. Because Golang binaries statically compile a massive amount of standard library code into the final executable, the automated rules focused on these common libraries rather than the malicious logic itself, risking an unacceptably high false-positive rate in a production environment.

To construct a high-fidelity signature, the malware required manual reverse engineering. The FrostyGoop binary was decompiled using Ghidra, an advanced software reverse engineering framework. Because Ghidra natively struggles with the unique structure of Go binaries, specific configurations and tools (such as GoReSym) were integrated to properly reconstruct the function names and string tables. By filtering the extracted strings for ICS-specific terminology (e.g., "modbus"), critical Indicators of Compromise (IoCs) were discovered. Notably, the analysis revealed hardcoded references to a legitimate, open-source Modbus interaction repository ([\[github.com/rolfl/modbus\]\(https://github.com/rolfl/modbus\)](https://github.com/rolfl/modbus)) that the threat actor had lazily incorporated into the malware's core logic.

To optimize the rule creation process and demonstrate a modern SOC workflow, the extracted string data was compiled into a CSV format and processed using a Large Language Model (LLM, specifically Gemini). The LLM was utilized to systematically refine the strings, cross-reference them with known programming paradigms, and assist in the rapid, automated generation of the YARA signature. The resulting rule was subsequently validated against community intelligence repositories, such as Valhalla, confirming alignment with existing open-source rulesets.

5.1.2 The Evasion Problem: Limitations of Network-Based Inspection.

With a robust YARA signature generated, the rules were deployed onto the laboratory's network-based sensors. The evaluation utilized both a high-end commercial solution (Nozomi Networks) and an open-source YARA-based IDS (YAIDS). The attacker machine, simulating a compromised pivot point

in the DMZ, then attempted to transfer the FrostyGoop executable to the OT network via standard HTTP protocols (wget).

The experimental results were unexpectedly poor. Despite possessing the exact signature, the network sensors frequently failed to trigger an alert during the active transfer of the malware. Extensive analysis of these failures revealed severe, systemic vulnerabilities in network-based detection, fundamentally related to the mechanics of Deep Packet Inspection (DPI) and high-speed network parsing.

Commercial NIDS, such as Nozomi Networks, are designed to monitor massive volumes of industrial traffic without introducing latency. To achieve this performance, they must make architectural compromises. One critical compromise is the implementation of hardcoded depth limits for file extraction. A NIDS will typically extract and scan only the first segment of an incoming file to check for magic bytes, headers, and early strings. If the entire file is larger than this limit, the NIDS discards the rest of the payload to prevent memory exhaustion and maintain network throughput.

The FrostyGoop Golang executable is inherently massive, weighing in at approximately 7.8 MB. Because the malicious strings (such as the rolfl/modbus reference) were buried deep within the binary structure, far beyond the NIDS inspection threshold, the YARA scan returned a false negative. While some IT firewalls (like Fortinet) can bypass this by routing large files to an external sandbox (FortiSandbox), such integrations are often absent in highly isolated OT environments.

Modern threat actors are acutely aware of this DPI limitation. They actively exploit it using an evasion technique known as Binary Padding. By artificially inflating a tiny, 100 KB malicious script with megabytes of null bytes (0x00) or junk data, attackers can easily push their malicious payload past the NIDS inspection window, rendering network-based signature scanning useless.

The fundamental architecture of YARA is designed to scan static, contiguous files residing on a physical disk. However, when deployed on a stateless packet inspector like YAIDS, the engine is forced to evaluate network traffic on a packet-by-packet basis.

When a 7.8 MB executable is transmitted over TCP/IP, it is fragmented into thousands of individual packets. Because the YARA rule was looking for a combination of strings, the condition frequently failed. For example, String A might arrive in packet 105, while String B arrives in packet 106. A stateless IDS cannot correlate them. Detecting complex patterns in transit requires computationally expensive stateful stream reassembly—a mechanism that holds packets in RAM, reconstructs the entire file, and then scans it. While tools like Suricata can perform this, the memory overhead is often prohibitive for continuous OT monitoring.

Executing complex YARA rules (especially those utilizing regular expressions or multiple string combinations) across every incoming packet is highly CPU-intensive. During the experimental unthrottled file transfers from the DMZ, the sheer velocity of the incoming packets rapidly saturated the processing queue of the YAIDS sensor.

When a software queue overflows, the Linux kernel intervenes to prevent a system crash by silently dropping incoming packets. Consequently, the malware successfully reached the target HMI endpoint because the specific packets containing the malicious strings were dropped by the overloaded IDS before they could even be analyzed. Successful network detection was only achieved when the transfer speed was artificially and unrealistically throttled (e.g., utilizing `wget --limit-rate=50k`).

These empirical tests provided a crucial conclusion: relying solely on network-based YARA scanning or commercial DPI is inadequate against modern, heavily padded, or fragmented ICS malware. To achieve reliable detection, the defensive perimeter must be shifted. The YARA engine must be moved away from the volatile network stream and placed directly onto the asset being attacked, analyzing the file only after it has been fully reassembled on the local disk.

5.2 Shifting the Paradigm: EDRs and Endpoint File Scanning

The empirical failures of Deep Packet Inspection (DPI) and stateless NIDS highlighted in the previous section necessitate a strategic pivot in the OT defense architecture. Relying exclusively on perimeter network security creates a critical single point of failure, particularly against modern adversaries employing evasion techniques like binary padding. To counter this, the defensive strategy must embrace the principle of Defense in Depth, shifting the detection engine to the final destination of the malicious payload: the endpoint itself.

5.2.1 Bypassing Network Limitations

By deploying an Endpoint Detection and Response (EDR) platform, the physical and computational limitations of network inspection are entirely bypassed. When a NIDS struggles with TCP stream reassembly, fragmentation, and packet dropping under heavy traffic loads, an EDR simply waits. It relies on the endpoint's own operating system kernel to successfully receive, reassemble, and write the network packets into a contiguous file on the local storage.

Once the executable is static and fully reconstructed on the disk, a signature-matching engine like YARA can analyze the entire binary. Operating at the endpoint level removes the strict processing time constraints and size depth limits imposed by high-speed network sensors; the EDR can take the

necessary milliseconds to scan a massive 7.8 MB padded executable from beginning to end, ensuring that deep indicators (such as the Golang library strings in FrostyGoop) are reliably matched.

5.2.2 Implementation via Velociraptor

To implement this paradigm shift within the Digital Twin, the Velociraptor EDR platform was utilized. Lightweight Velociraptor agents were deployed across the critical SCADA endpoints. Instead of relying on periodic, scheduled scans that leave wide windows of vulnerability, the deployment leveraged Velociraptor's asynchronous capabilities.

By creating specific `CLIENT_EVENT` artifacts, the local agent was configured to hook into the operating system's file monitoring APIs. This allowed the EDR to perform automated, real-time YARA scanning exclusively on newly created or modified files.

During the validation testing, when the padded FrostyGoop payload successfully evaded the YAIDS network sensor and was downloaded onto the target HMI, the local Velociraptor agent instantly detected the file creation event. The agent analyzed the fully reconstructed binary, successfully matched the deep YARA signature, and triggered a critical alert. This empirical test definitively proved that complex, deep-searching YARA rules must reside on the endpoint rather than on the network.

5.2.3 The Inherent Limitation of Signature-Based Detection

While this endpoint-centric approach successfully resolves the physical limitations of network fragmentation, it remains fundamentally constrained by its underlying methodology: signature-based detection. Tools like YARA are exceptionally effective at identifying known malware families or new variants that share recognizable, pre-analyzed code similarities. However, this reactive approach possesses a critical blind spot: it relies entirely on prior intelligence. When confronted with bespoke, "zero-day" malware engineered specifically for a single facility, or an ad-hoc script never seen before in the wild, static file scanning inevitably fails because no pre-existing signature exists to trigger the alert.

Furthermore, sophisticated adversaries increasingly utilize "Living off the Land" (LotL) techniques, leveraging legitimate, pre-installed administrative tools to execute their attacks without writing new, recognizable malware to the disk. The "Stuxnet-like" False Data Injection (FDI) attack demonstrated in Chapter 4 exemplifies this evasion. It did not utilize a compiled malware executable; instead, it leveraged native Python libraries and standard Linux firewall utilities (iptables) to silently manipulate Modbus traffic directly in the network stack.

This dual evasion—employing unknown, custom logic via legitimate, fileless execution—renders both NIDS and traditional file-scanning EDRs completely blind. Addressing this deception requires the ultimate evolution of the proposed defensive model: transitioning from static file scanning to active Semantic Memory Hunting.

5.3 The Fileless Threat: Detecting FDI in SCADA Memory

The endpoint-centric file scanning architecture effectively mitigates traditional, compiled ICS malware. However, it exposes a fundamental semantic gap in current defense paradigms when confronted with "Living off the Land" (LotL) execution and False Data Injection (FDI) attacks. If an adversary compromises an endpoint and utilizes pre-existing, legitimate system administration tools to manipulate network telemetry, traditional Endpoint Detection and Response (EDR) agents will not generate an alert, as no malicious binary is ever written to disk.

5.3.1 The State of the Art and the "Sensory Blindness" Dilemma.

Currently, the industry relies on a bifurcated approach to OT security: passive Network Intrusion Detection Systems (NIDS) analyze network traffic for anomalies, while EDRs scan endpoint storage for known malware signatures.

This bifurcation fails catastrophically during sophisticated UI Spoofing or FDI attacks. Consider a scenario where an attacker intercepts telemetry locally on the HMI's network stack, overwriting a critical process variable. The operator's display is frozen at a safe, nominal value, while the physical controller (PLC) is driven to a critical or destructive state.

In this scenario:

- The NIDS is blind because the packet manipulation occurs locally within the endpoint's internal routing, bypassing the network perimeter entirely.
- The traditional EDR is blind because the scripts executing the manipulation utilize native OS utilities, failing to trigger static malware signatures.

The result is absolute "Sensory Blindness". The physical process is actively compromised, yet traditional IT and OT security dashboards report a secure and stable status.

5.3.2 The Innovation: Semantic Memory Hunting.

To resolve this critical blind spot, this thesis proposes a novel defensive paradigm: Semantic Memory Hunting. The core concept is to transcend the traditional use of EDR platforms. Instead of utilizing

tools like YARA and Velociraptor solely to search for malicious code syntax, these tools are radically repurposed to extract and validate process data-the actual physical variables-directly from the volatile memory (RAM) of the industrial supervisory applications.

By doing so, the EDR is transformed from a generic IT security tool into a "Physics-Aware" sensor.

The defensive objective fundamentally shifts from Code Validation to State Validation.

The methodology for Semantic Memory Hunting relies on a real-time correlation engine executing a deterministic sequence of operations:

1. **Physical State Acquisition (Polling):** An automated, out-of-band request (via a custom Python script) directly polls the physical controller (the PLC) using standard industrial protocols to acquire the true, current state of the critical process variables.
2. **Dynamic Signature Generation:** The acquired numerical values from the PLC are dynamically converted into their exact byte-level representation (e.g., IEEE 754 hexadecimal floating-point format). This data is then utilized to programmatically generate a highly specific, custom YARA rule on the fly.
3. **Memory Execution and Validation:** The EDR agent launches a targeted artifact (a custom VQL query in Velociraptor) utilizing this dynamically generated YARA signature to scan the volatile memory (JVM heap) of the SCADA application process.
4. **Correlation and Alerting:** The memory hunt acts as a verification mechanism to ensure that the true physical values retrieved from the PLC are actively present and utilized within the HMI's operational memory. If the EDR scan fails to find these specific values (or detects an anomalous discrepancy in the expected memory hits), it confirms that the HMI is rendering manipulated data, triggering an immediate False Data Injection alert.

If the extracted RAM value deviates from the actual physical state of the PLC, the EDR immediately triggers a critical FDI alert. This mechanism effectively unmask the attack by demonstrating that the HMI's memory has been semantically compromised and no longer reflects physical reality, bypassing the attacker's network-layer deception entirely.

5.4 Reverse Engineering the JVM Heap (Tomcat/ScadaBR)

To execute Semantic Memory Hunting effectively, the EDR must know exactly what to look for within the vast and volatile memory space of an active application. ScadaBR operates within an Apache

Tomcat Java Virtual Machine (JVM). Therefore, the critical physical values read from the Modbus PLCs are dynamically allocated and stored as Java objects within the JVM heap.

The reverse engineering phase aimed to definitively identify how these process variables (referred to as Tag Values or Data Points in SCADA terminology) are mapped and structured in memory, effectively translating a physical Modbus state into a huntable hexadecimal signature.

5.4.1 Memory Dumping and Identification

Initial attempts to isolate the variables involved forcing the JVM to output its memory state using standard Java debugging tools (such as jmap and hprof). However, analyzing these massive dumps with tools like Eclipse Memory Analyzer (MAT) proved overly complex and inefficient for real-time hunting.

To overcome this, a custom memory extraction script was utilized to directly dump the raw virtual memory segments of the active Tomcat process from the Linux `/proc/[PID]/mem` interface. A controlled experiment was conducted within the Digital Twin: a specific Modbus Input Register (e.g., Register ID 13, representing the pasteurization temperature) was forced to a highly specific, recognizable floating-point value (e.g., 99.0).

Simultaneously, a Python script was used to convert this target floating-point value into its equivalent IEEE 754 double-precision (64-bit) little-endian hexadecimal representation:

```
struct.pack("<d", 99.0): 00 00 00 00 00 C0 58 40
```

By deploying a preliminary YARA scan against the raw memory dump using this specific byte sequence, multiple matches were discovered.

5.4.2 Overcoming the Garbage Collection "Phantom" Matches

A significant challenge emerged during this identification phase: executing a targeted YARA scan often returned a variable number of matches (typically between 1 and 4) for the same numerical value. Furthermore, even after the physical value had changed on the PLC, "phantom" matches of the old value persisted in memory.

This behavior is a fundamental characteristic of the Java Garbage Collector (GC). When ScadaBR updates a Data Point, it does not overwrite the old variable in place; it allocates a new Java object and abandons the old one. The old values remain dormant in the JVM heap as "phantoms" until the GC algorithm decides to reclaim that specific memory segment.

To isolate the "live" object—the actual value currently rendered on the HMI dashboard—the Tomcat process was forced to execute a manual Garbage Collection routine (`sudo jcmd <PID> GC.run`). This flushed the unreferenced phantom objects, leaving behind only the actively utilized memory structure.

5.4.3 The Semantic Signature Structure

Analyzing the offset of this active, surviving object revealed a highly consistent, structured pattern in the JVM heap. The active `PointValue` object maintained by ScadaBR consists of specific identifiers that correlate directly with its backend SQL database.

The reverse engineering determined that a reliable, high-fidelity YARA signature for a "live" Modbus variable in ScadaBR requires a specific sequence:

1. The "Live" Marker: The active object is consistently preceded by an 8-byte sequence of FF (FF FF FF FF FF FF FF), which replaces an older timestamp field when the object is fully instantiated for the UI.
2. The Process Value: Immediately following the marker is the 8-byte hexadecimal representation of the floating-point value.
3. The Memory Gap: A predictable padding/offset gap (exactly 52 bytes) separates the value from its database identifier.
4. The Data Point ID: The sequence concludes with the 4-byte little-endian representation of the specific Data Point ID (e.g., ID 24) assigned by the ScadaBR database.

By identifying this exact structural layout, it became possible to construct a dynamic, highly targeted YARA rule.

With the semantic mapping of the JVM heap completed, the final step was to automate this correlation engine using Velociraptor and gRPC, creating the ultimate FDI detection artifact.

5.5 Developing the Custom VQL Artifact: Correlating RAM Data with PLC States

With the exact semantic structure of the SCADA variables identified within the JVM heap, the final objective was to engineer a scalable, automated detection pipeline capable of continuously auditing the integrity of the HMI in real-time. To achieve this, a dual-component architecture was developed, integrating an out-of-band Python orchestration script with a custom Velociraptor Query Language (VQL) artifact deployed on the endpoint.

To demonstrate the baseline capabilities of the proposed Semantic Memory Hunting mechanism, the experimental deployment was initially configured to simultaneously monitor 100 distinct Modbus Input Registers, simulating a highly critical industrial control segment.

5.5.1 Scalability and Batch Execution Strategy

While the initial deployment successfully monitored 100 registers in a single execution, real-world SCADA environments routinely manage hundreds or thousands of critical data points. Scaling the Semantic Memory Hunting architecture to accommodate this volume required a structured execution strategy. Generating a single, monolithic YARA rule containing 500 or more signatures would introduce severe performance degradation, resulting in massive VQL payload sizes, excessive rule compilation overhead, and dangerous CPU spikes on the fragile ICS endpoint.

To resolve this, the architecture implements a batched execution model. The Python orchestration script is programmed to establish a hard limit, capping the maximum number of evaluated registers at 100 per single Velociraptor hunt. When the total number of monitored sensors exceeds this threshold, the script automatically partitions the Modbus polling results into smaller, discrete chunks (e.g., 500 registers are divided into 5 independent batches of 100). The gRPC API then sequentially triggers a series of batched VQL artifacts. This chunking strategy ensures that the EDR agent processes the memory evaluation in manageable, highly optimized bursts, preserving the system's strict Operations Security (Ops/Sec) limits and guaranteeing zero impact on the continuous availability of the industrial HMI.

5.5.2 Out-of-Band Polling and Dynamic Signature Generation

The first component of the architecture is the external Python monitoring engine. Crucially, this script does not execute on the HMI, which is assumed to be potentially compromised at the network stack level (as demonstrated in Chapter 4). Instead, it operates out-of-band, communicating directly with the PLC via Modbus TCP (Port 502, Function Code 4) to establish the absolute "ground truth" of the physical process.

During each polling cycle, the engine retrieves the values of the monitored registers. Rather than logging these values, the script programmatically translates them into custom YARA strings based on the semantic mapping established in the previous section. For each register, the script:

1. Converts the floating-point numerical value into its 64-bit IEEE 754 little-endian hexadecimal equivalent.

2. Converts the corresponding ScadaBR Data Point ID (e.g., ID 24) into a 32-bit hexadecimal format.
3. Assembles the final, surgical signature utilizing the previously discovered "live object" marker (FF FF FF FF FF FF FF FF) and the exact 52-byte memory offset gap ([52-52]).

This process results in the dynamic generation of unique, highly specific memory signatures on the fly.

5.5.3 The Custom VQL Artifact Execution

Deploying these dynamic YARA rules manually via a web interface is operationally unfeasible. To achieve enterprise-grade automation, the Python engine leverages the pyvelociraptor library to communicate directly with the central Velociraptor Server via the gRPC protocol.

The script packages the dynamically generated YARA signatures into a structured API request. It injects these signatures as environmental variables (`env=dict(...)`) and triggers a targeted memory collection task specifically aimed at the compromised SCADA HMI client. This asynchronous, API-driven architecture ensures that the memory hunt is constantly updated with the real-time physical state of the PLC.

Upon receiving the API trigger, the Velociraptor agent on the HMI executes a highly specialized CLIENT artifact (`Custom.Artifact.memory.scadaBR.FinalSignature.Variables...`).

To strictly enforce the operational safety requirements of an OT environment, the VQL query does not blindly scan the entire operating system's memory. Instead, it executes a highly optimized two-step process:

- **Process Targeting:** The VQL engine queries the operating system's process list (`pslist()`), specifically filtering for the active Java process executing the SCADA application. This surgical targeting limits the scanning scope exclusively to the relevant JVM memory space.
- **In-Memory YARA Execution:** The artifact dynamically constructs a monolithic YARA rule containing the batch of injected variables wrapped in an `any of them` condition. It then executes the `proc_yara()` plugin against the isolated Tomcat PID.

The artifact streams the results back to the central server.

The detection logic of Semantic Memory Hunting relies on absence as an Indicator of Compromise. The YARA rules were generated using the true physical data currently existing on the PLC; the VQL artifact must find those exact signatures in the HMI's memory if the system is operating normally. If an attacker has executed a False Data Injection-altering the network telemetry so that the HMI receives and renders spoofed data-the exact hexadecimal structures generated by the Python script will not exist

in the JVM heap. Consequently, the VQL query will return zero matches (or a significantly reduced number of matches) for the compromised registers.

By detecting this semantic mismatch between the physical truth (polled via Python) and the digital representation (scanned via VQL), the proposed architecture successfully detects the FDI attack, overcoming the sensory blindness imposed by the adversary.

6. Experimental Results and Performance Evaluation

To validate the theoretical defense mechanisms and architectural paradigms proposed in the previous chapters, a comprehensive series of empirical evaluations was conducted within the Digital Twin environment. The testing methodology was designed to measure two distinct but equally critical dimensions of Operational Technology security: detection efficacy and system performance impact. This chapter details the quantitative results of these experiments, transitioning from a baseline evaluation of traditional signature matching against historical ICS malware to the validation of the novel Semantic Memory Hunting mechanism against fileless False Data Injection (FDI) attacks.

6.1 Comparative Efficacy on Known ICS Malware Families

As established in Chapter 5, passive Network Intrusion Detection Systems (NIDS) operating at the IT/OT boundary face severe structural challenges when evaluating large, fragmented, or padded malicious payloads in transit. To quantitatively validate this operational gap and demonstrate the necessity of endpoint visibility, a comparative benchmark was executed.

A corpus of ten highly significant malware families—encompassing historic state-sponsored IT/OT crossover threats (e.g., Stuxnet, Havex, BlackEnergy) and modern, ICS-specific ransomware and attack frameworks (e.g., EKANS, Incontroller)—was selected.

These payloads were transferred into the experimental network and evaluated against four distinct detection methodologies:

1. NOZOMI Networks: A leading commercial, appliance-based ICS network sensor.
2. YAIDS: An open-source, stateless network intrusion detection system natively running YARA rules against individual TCP packets.
3. YARA + Python: A host-based execution model where local Python scripts trigger YARA scans on static files resting on the endpoint's disk.

4. Velociraptor + YARA: The enterprise-grade Endpoint Detection and Response (EDR) platform executing asynchronous file system and in-memory YARA hunting.

The detection results for each malware family across the four platforms are summarized in Table 6.1.

Table 6.1: Detection Efficacy Benchmark across Network and Endpoint Platforms

ICS Malware Family	NOZOMI (Network)	YAIDS (Network)	YARA+PYTHON (Endpoint)	Velociraptor+YARA (Endpoint)
Stuxnet	✗	✗	✓	✓
Havex	✓	✗	✓	✓
BlackEnergy	✗	✗	✓	✓
Industroyer	✗	✗	✓	✓
Triton	✗	✗	✓	✓
WannaCry	✓	✗	✓	✓
NotPetya	✗	✗	✓	✓
EKANS	✓	✗	✓	✓
Incontroller	✗	✗	✓	✓
Industroyer2	✓	✗	✓	✓

(Legend: ✓ = Successful Detection / Prevention; ✗ = Failed Detection / False Negative)

The empirical data clearly illustrates the systemic vulnerabilities of relying exclusively on perimeter inspection. The open-source stateless engine, YAIDS, exhibited a 100% failure rate across the benchmark. Because YAIDS cannot perform TCP stream reassembly, complex YARA rules containing multifaceted conditions (e.g., matching a Modbus string and a specific file header) consistently failed due to the network packet fragmentation of the executables.

The commercial network sensor, NOZOMI, demonstrated partial efficacy, successfully flagging threats like Havex, WannaCry, EKANS, and Industroyer2. These detections are largely attributed to the appliance identifying known, unencrypted Command and Control (C2) network signatures or smaller,

unpadded initial dropper stages. However, NOZOMI completely failed to detect massive, heavily padded, or highly obfuscated payloads (such as Stuxnet, Triton, and BlackEnergy) during network transit. As analyzed in Section 5.1, this is a direct consequence of the hardcoded Deep Packet Inspection (DPI) depth limits; the appliance simply discards the remainder of large files to prevent internal RAM exhaustion, allowing the core malicious logic to pass unnoticed.

Conversely, both endpoint-based methodologies (YARA+Python and Velociraptor) achieved a 100% detection rate across all tested samples. By shifting the detection engine to the target host, these platforms bypass network fragmentation and DPI size limitations entirely. The host operating system reconstructs the executable into a contiguous file on the disk or allocates it in virtual memory, allowing the YARA engine to parse the entire binary structure without stringent temporal or size constraints.

These results definitively prove that robust, high-fidelity YARA scanning—essential for detecting sophisticated ICS threats—requires continuous endpoint visibility.

The Operational Superiority of the EDR Architecture

While both endpoint-based methodologies successfully detected the known malware payloads, establishing parity in raw static detection efficacy, they differ drastically in operational viability for an Industrial Control System (ICS) environment.

The custom YARA + Python implementation, although functionally effective in a controlled laboratory, presents severe deployment and scalability challenges. Relying on custom automation requires installing the Python interpreter, the core YARA engine binaries, and various dependency libraries directly onto the critical SCADA endpoints. This heavily increases the system footprint, introduces substantial management overhead ("heavy load of work"), and frequently violates strict OT compliance regulations regarding unauthorized software installations. Furthermore, this ad-hoc approach lacks a central management server, rendering it unfeasible to orchestrate widespread threat hunts, update signatures, or aggregate telemetry across a distributed industrial plant floor.

Conversely, the Velociraptor EDR provides a monolithic, enterprise-grade architecture. It requires only the deployment of a single, self-contained, statically compiled lightweight agent, completely eliminating the need for external runtime dependencies or local Python installations. Beyond the drastically simplified deployment, Velociraptor unlocks advanced Digital Forensics and Incident Response (DFIR) capabilities that isolated Python scripts cannot safely or efficiently replicate. These include natively embedded YARA engines, real-time asynchronous event triggering (CLIENT_EVENT), and strict client-side CPU throttling (Ops/Sec limits) to guarantee the continuous availability of the industrial process.

Most importantly, Velociraptor's client-server gRPC architecture and its advanced query language (VQL) permit the safe interrogation of volatile process memory (`proc_yara()`). Because of these critical operational advantages, Velociraptor was selected not merely as the superior choice for static malware detection, but as the mandatory foundational engine required to execute the advanced Semantic Memory Hunting mechanism detailed in the following sections.

6.2 Validation of the Semantic Memory Hunting Mechanism

With the operational superiority of the Endpoint Detection and Response (EDR) platform established, the experimental evaluation transitioned to validating the novel Semantic Memory Hunting mechanism. The objective was to empirically prove that the out-of-band Python polling script and the custom Velociraptor VQL artifact could successfully and reliably detect the fileless False Data Injection (FDI) attack.

6.2.1 Baseline Verification and False Positive Rate (FPR)

A critical metric for any detection system operating within an Industrial Control System (ICS) environment is the False Positive Rate (FPR). An elevated FPR can lead to alarm fatigue or trigger unnecessary, costly emergency plant shutdowns.

The first phase of empirical testing evaluated the architecture under nominal operating conditions. The Python monitoring engine continuously polled the Modbus registers and dynamically generated the corresponding YARA signatures, which were subsequently executed by the Velociraptor agent on the SCADA HMI.

Thanks to the precise semantic mapping and the isolation of the "live" Java objects detailed in Section 5.4, the continuous memory scans yielded a 0% False Positive Rate. The custom VQL artifact consistently located the exact, dynamically generated hexadecimal signatures within the JVM heap. The engine successfully ignored unreferenced "phantom" memory blocks pending Java Garbage Collection, confirming that the digital representation on the HMI perfectly correlated with the physical reality of the PLCs.

6.2.2 FDI Attack Detection Validation

The second phase of the evaluation tested the architecture against an active threat. The "Stuxnet-like" UI Spoofing attack was executed: the local iptables rules and the malicious Python script successfully intercepted Modbus responses on the HMI. The physical PLC was driven to a critical state (e.g. 90 C), but the HMI application was forced to render a spoofed, nominal value (e.g. 50 C).

Upon execution of the Semantic Memory Hunting cycle during the active attack, the system operated exactly as designed:

1. The out-of-band Python script accurately bypassed the network deception, retrieving the true physical value 90.0 directly from the PLC.
2. The script generated the specific YARA signature for this critical value, strictly bound to its Modbus Register ID.
3. The Velociraptor agent executed the scan against the Tomcat process memory, attempting to validate the presence of this specific physical state.

Because the HMI had been deceived and was only allocating the spoofed 50.0 value within its active memory structures, the YARA scan returned zero matches for the targeted register. This immediate, deterministic failure to find the true physical state within the digital supervisory memory successfully triggered the critical FDI alert.

The empirical results validated the core thesis hypothesis: the "Physics-Aware" EDR paradigm successfully detected a sophisticated, fileless UI Spoofing attack that traditional network-based monitoring systems and static file scanners had entirely missed.

6.3 System Impact Analysis on ICS Endpoints

While the Semantic Memory Hunting architecture successfully demonstrated high-fidelity detection capabilities against sophisticated FDI attacks, deploying active scanning mechanisms in an Operational Technology environment introduces a critical operational risk. The fundamental mandate of ICS cybersecurity is the preservation of Availability and physical safety. If an Endpoint Detection and Response (EDR) agent consumes excessive CPU cycles or exhausts system memory during a hunt, it can cause the SCADA application to crash or introduce unacceptable latency into the real-time control loop, effectively executing a self-inflicted Denial of Service (DoS).

To comprehensively evaluate the performance footprint of the Velociraptor agent during active memory scanning, a rigorous system impact analysis was conducted.

6.3.1 Experimental Constraints and Baseline Configuration

The performance benchmarks were executed directly on the SCADA HMI endpoint. To accurately contextualize the telemetry, the hardware specifications of the virtualized environment must be considered: the Ubuntu Linux HMI was provisioned with 4 vCPUs and 4 GB of RAM, with disk I/O

strictly throttled at the hypervisor level to 5 MB/s to simulate the constrained hardware often found in legacy industrial deployments.

System telemetry was continuously captured utilizing the Linux pidstat utility. To ensure analytical accuracy, the telemetry was segmented to isolate the metrics during the system's baseline resting state (Idle Phase) from the metrics strictly recorded during the active memory scan (Active Hunt Phase).

The evaluation measured the resource consumption of the Velociraptor client across two distinct operational scenarios:

1. Semantic Hunt (Unthrottled): The VQL memory artifact is executed with maximum priority, utilizing all available system resources to complete the scan as rapidly as possible.
2. Semantic Hunt (Throttled): The VQL artifact is executed with a strict Velociraptor client-side Operations/Second (Ops/Sec) limit enforced, artificially slowing down the scan to preserve CPU cycles for the SCADA application.

The quantitative results of these scenarios, clearly distinguishing between idle and active phases, are summarized in Table 6.2.

Table 6.2: Velociraptor EDR Resource Consumption on SCADA HMI (4 vCPU, 4GB RAM)

Execution Scenario	Active Scan Time	Avg CPU (Idle Phase)	Avg CPU (Active Phase)	Peak CPU	Avg RAM (Idle Phase)	Avg RAM (Active Phase)	Peak RAM
Semantic Hunt (Unthrottled)	~ 16 seconds	< 1.00 %	~ 85.00 %	137.00 %	81.00 MB	105.00 MB	125.00 MB
Semantic Hunt (Throttled)	> 11 minutes	< 1.00 %	~ 4.37 %	108.00 %	90.00 MB	101.53 MB	115.00 MB

6.3.2 Analysis of Memory Footprint (RAM)

The empirical data indicates that the memory footprint of the Velociraptor agent is highly optimized and exceptionally stable. During the absolute peak of an unthrottled memory hunt, the agent allocated a maximum of roughly 125 MB of RAM. On an industrial endpoint provisioned with 4 GB (4096 MB) of total memory, the active EDR scan consumes a negligible ~3% of the available system RAM. This minimal footprint guarantees that Semantic Memory Hunting can be continuously executed without risking Out-Of-Memory (OOM) exceptions or starving the active Tomcat/ScadaBR processes of vital memory resources.

6.3.3 Analysis of Computational Overhead (CPU) and Throttling

In Linux environments, a CPU usage metric of 100% equates to the full saturation of a single CPU core. Because the HMI possesses 4 vCPUs, the absolute theoretical maximum is 400%.

During the Unthrottled execution scenario (illustrated in Figure 6.1), the Velociraptor engine aggressively attempted to read the JVM heap. Isolating the telemetry exclusively to the active scanning phase reveals a severe computational burden: the CPU usage averaged roughly 85.00% continuously, with a brutal spike reaching 137.00%. While the scan completed rapidly (in approximately 16 seconds), fully saturating one core and heavily loading a second core is a dangerous anti-pattern in critical ICS environments, as it risks interrupting the HMI's real-time polling cycles and causing unacceptable latency.

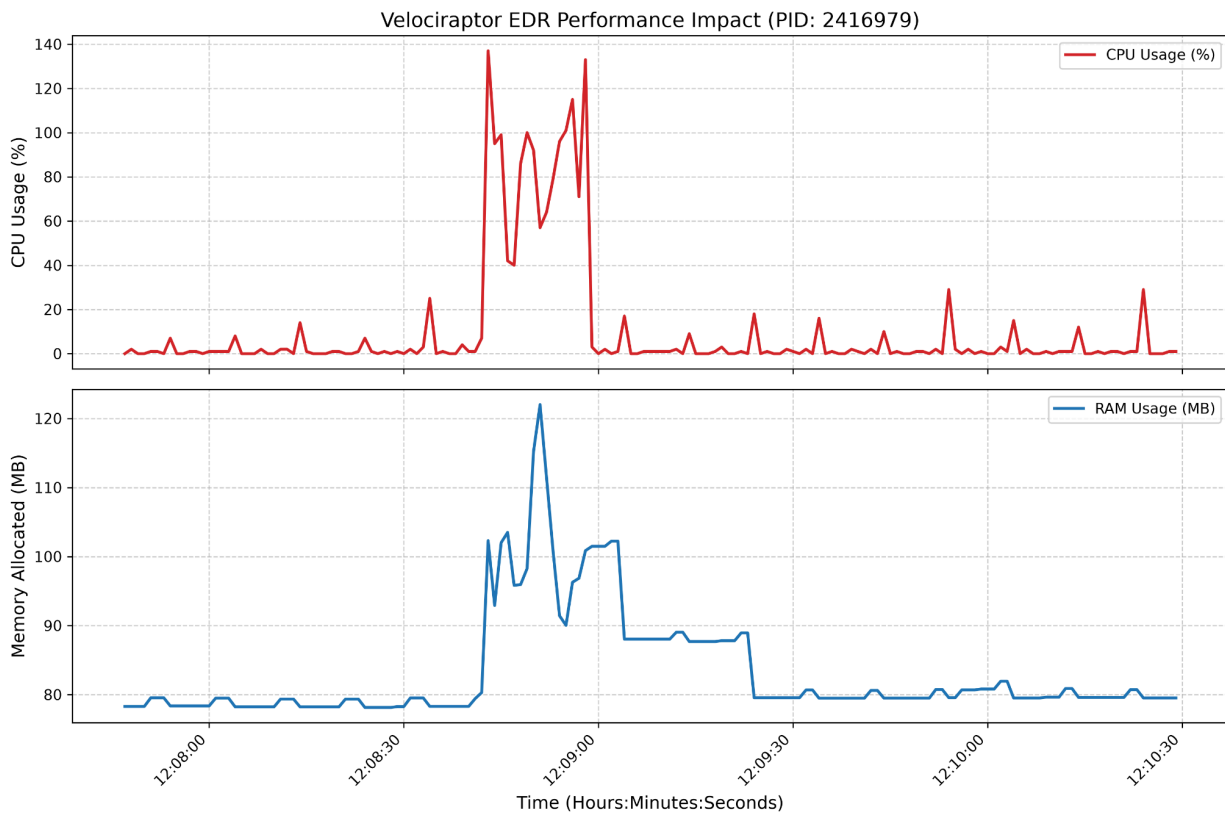


Figure 6.1: UnThrottled Velociraptor Agent

To resolve this operational hazard, the Throttled execution scenario (illustrated in Figure 6.2) was implemented. By enforcing a strict Ops/Sec limit via the Velociraptor management console, the agent was forced to introduce micro-pauses during the memory scan.

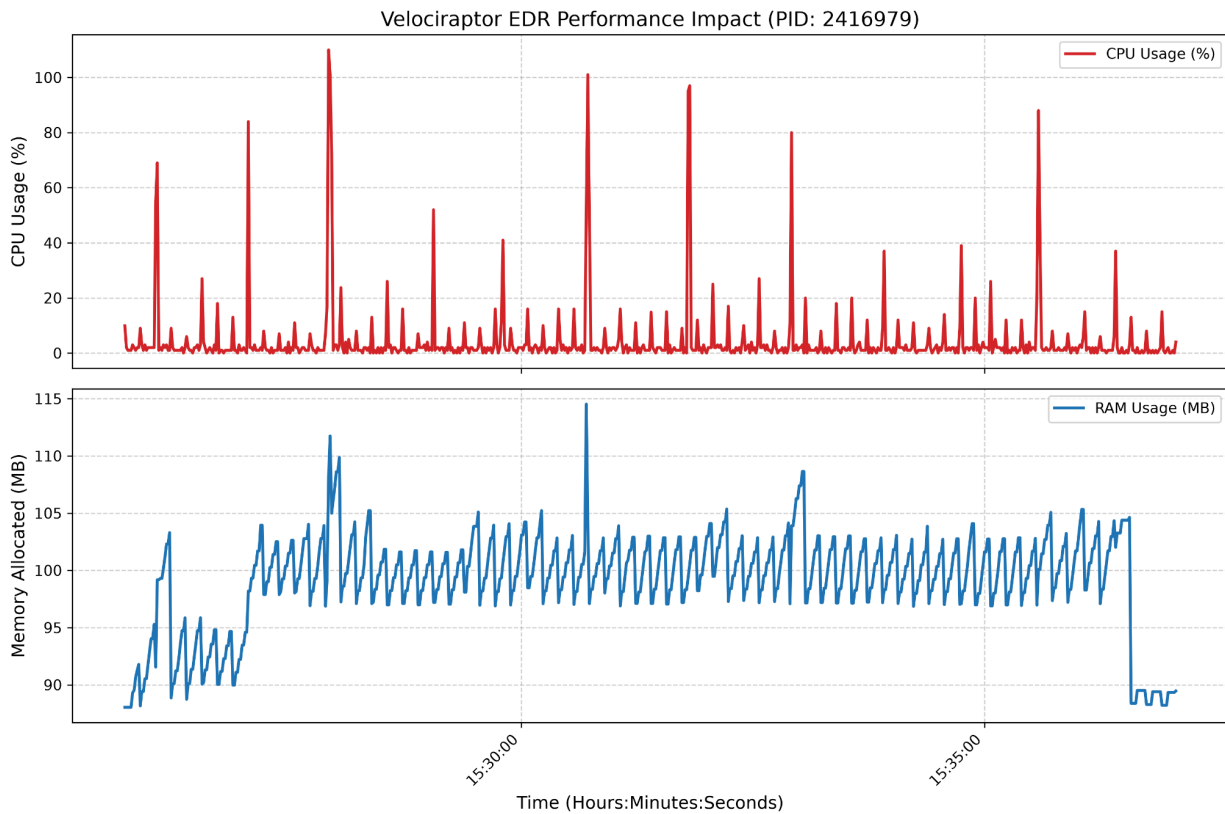


Figure 6.2: Throttled Velociraptor Agent

As clearly demonstrated in the telemetry, the throttling mechanism drastically alters the system impact. Instead of a massive, contiguous CPU drain, the computational load is transformed into a series of brief, highly manageable micro-bursts separated by extended periods of near-zero utilization. Consequently, the total execution time stretches from 16 seconds to over 11 minutes, but the active average CPU usage drops catastrophically from 85.00% to a mere 4.37%.

This specific configuration proves that Semantic Memory Hunting is entirely viable in production ICS environments. However, it is crucial to emphasize that the 11-minute execution time observed in the throttled scenario is the direct result of applying deliberately strict limits to the Velociraptor agent for experimental validation.

In a real-world production environment, this parameter is not a universal constant. Rather, it represents a highly configurable variable that must be meticulously tuned on a case-by-case basis. Security engineers must actively manage the inherent trade-off between detection velocity and hardware utilization. A more robust, modern SCADA server could safely sustain a higher Ops/Sec limit, thereby compressing the detection window to a matter of minutes or seconds. Conversely, a highly critical or legacy endpoint might necessitate even stricter throttling to absolutely guarantee the uninterrupted execution of the industrial control loop. Ultimately, the EDR configuration must be dynamically

tailored to the specific hardware capabilities, the criticality of the monitored variables, and the operational risk tolerance of the individual industrial facility.

7. Limitations and Conclusions

7.1 Limitations of the Experimental Infrastructure and the Proposed Detection Artifact

While the empirical results successfully validated the core hypothesis of this thesis, it is necessary to contextualize the findings within the boundaries of the experimental setup. Acknowledging these limitations provides a realistic assessment of the proposed architecture and highlights areas requiring further engineering before enterprise-wide deployment.

First, the experimental environment, although a highly realistic "Digital Twin," remains a simulation. The network traffic volume, the complexity of the Active Directory domain, and the background noise generated by the simulated endpoints cannot perfectly replicate the chaotic environment of a massive, multi-site industrial facility.

Second, a critical limitation resides in the scalability of the Semantic Memory Hunting artifact. As analyzed in Section 6.3, the system successfully monitored batches of 100 Modbus registers while maintaining a sustainable CPU load. However, modern critical infrastructure facilities (such as power generation plants or vast chemical refineries) routinely process tens or hundreds of thousands of concurrent Data Points. Scaling the Python polling script and the Velociraptor VQL queries to accommodate this magnitude would inherently lengthen the batched execution cycles. This could potentially extend the detection window from minutes to hours, leaving the system temporarily exposed to rapid, destructive actuations before the memory scan completes its cycle.

Finally, the current implementation of the detection artifact is structurally bound to the specific technological stack of the laboratory. The dynamic YARA signature generation relies on the exact memory offset gap (52 bytes) and the specific hexadecimal representation utilized by the Apache Tomcat JVM hosting ScadaBR. Furthermore, the out-of-band polling script is hardcoded for the Modbus TCP protocol. Consequently, deploying this exact artifact on a different SCADA platform (e.g., Siemens WinCC or Inductive Automation Ignition) or utilizing a different industrial protocol would require a complete re-execution of the reverse engineering phase.

7.2 Conclusions on the Work Performed

The primary objective of this thesis was to investigate, adapt, and innovate advanced threat detection methodologies within converged IT/OT networks, moving beyond the inadequate paradigms of

traditional perimeter security. The research journey systematically addressed the evolving sophistication of industrial cyber threats.

The initial phases of the study conclusively demonstrated that traditional, network-based Deep Packet Inspection (DPI) solutions are structurally disadvantaged against modern malware. By analyzing the FrostyGoop payload, it was proven that threat actors can easily bypass Network Intrusion Detection Systems (NIDS) using evasion techniques like binary padding and TCP fragmentation. This empirical failure necessitated the first architectural paradigm shift: moving the YARA detection engine away from the volatile network stream and deploying Endpoint Detection and Response (EDR) agents to analyze fully reconstructed files on the host storage.

However, the threat modeling phase revealed a critical blind spot in standard EDR deployments. The execution of a localized, "Stuxnet-like" False Data Injection (FDI) attack demonstrated that adversaries can hijack native operating system tools to silently manipulate industrial telemetry without writing malicious binaries to disk. This fileless manipulation induces a state of "Sensory Blindness," rendering traditional IT security tools useless as the physical plant is driven toward failure while the operator dashboard reports nominal conditions.

To resolve this critical vulnerability, this thesis introduced and validated a novel defensive paradigm: Semantic Memory Hunting. By radically repurposing Velociraptor and YARA, the EDR was transformed into a "Physics-Aware" sensor. Instead of searching for malicious code, the architecture was engineered to extract process variables directly from the volatile memory (JVM heap) of the SCADA application and cross-reference them with the true physical state polled out-of-band from the PLCs.

The experimental validation proved highly successful. By identifying specific "live object" markers to overcome the statistical noise of the Java Garbage Collector, the architecture achieved a 0% False Positive Rate during nominal operations. During an active FDI attack, the system accurately and deterministically detected the semantic mismatch in RAM, successfully unmasking the UI Spoofing deception. Furthermore, through rigorous system impact profiling, it was demonstrated that by dynamically throttling the VQL execution limits, this advanced memory hunting can be performed safely on constrained ICS endpoints without disrupting the critical industrial control loop.

Ultimately, this thesis successfully bridges the semantic gap between Information Technology and Operational Technology. It proves that advanced DFIR tools, when creatively applied to the physical context of industrial processes, can effectively secure critical infrastructure against the most sophisticated, deceptive cyber-physical attacks.

7.3 Future Developments

The architecture developed in this research lays a robust foundation for the active defense of cyber-physical systems. To transition this prototype into a universally applicable industrial security framework, several key areas of future development are proposed:

1. **Protocol and Platform Agnosticism:** The immediate next step involves decoupling the Python orchestration script and the VQL artifact from Modbus TCP and ScadaBR. Future iterations should focus on developing modular parsers capable of polling native ICS protocols such as DNP3, IEC 61850, S7comm, and modern OPC UA architectures, accompanied by memory maps for industry-leading SCADA platforms.
2. **AI-Driven Automated Signature Engineering:** The most labor-intensive phase of Semantic Memory Hunting is the reverse engineering of the application's RAM to find the correct data offsets. Future research could integrate Large Language Models (LLMs) and Artificial Intelligence into the DFIR pipeline. An AI agent could automatically ingest standard industrial engineering files (such as .ST structured text programs or .L5X configuration files from PLCs), map the tags, and autonomously write the targeted VQL and YARA rules, eliminating the need for manual reverse engineering by human analysts.
3. **Closed-Loop Automated Incident Response (SOAR Integration):** Currently, the architecture operates as a highly advanced detection mechanism that generates a critical alert. The ultimate evolution of this system is to integrate it with a Security Orchestration, Automation, and Response (SOAR) platform. Upon detecting an FDI semantic mismatch, the EDR could automatically trigger a defensive playbook. This could involve instantly isolating the compromised HMI from the network via local firewall rules, or sending an out-of-band "Emergency Stop" command directly to the PLC, effectively closing the loop from detection to physical mitigation in a matter of milliseconds.

8. References

- [1] "OT IDS Network Monitoring in OT/ICS," Undercode Testing. [Online]. Available: <https://undercodetesting.com/ot-ids-network-monitoring-in-ot-ics/>.
- [2] "Top 10 Industrial IDS/IPS Systems for OT Protection," OT Ecosystem. [Online]. Available: <https://otecosystem.com/top-10-industrial-ids-ips-systems-for-ot-protection/>.
- [3] "Approaches to OT Network Monitoring," Claroty. [Online]. Available:

<https://claroty.com/blog/approaches-to-ot-network-monitoring/>.

[4] “Purdue Enterprise Reference Architecture,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Purdue_Enterprise_Reference_Architecture.

[5] “The Purdue Enterprise Reference Architecture: Strengthening IoT and OT Security in Industrial Environments,” Device Authority. [Online]. Available: <https://deviceauthority.com/the-purdue-enterprise-reference-architecture-strengthening-iot-and-ot-security-in-industrial-environments/>.

[6] “YAIDS,” GitHub. [Online]. Available: <https://github.com/wrayjustin/yaid>.

[7] “IEEE Xplore, Document 10145723.” [Online]. Available: <https://ieeexplore.ieee.org/document/10145723>.

[8] “YARA Rules: Complete Threat Hunting Guide,” M-Lab. [Online]. Available: <https://mlab.sh/blog/yara-rules-complete-threat-hunting-guide>.

[9] “YARA Rule Examples,” Phishing Tackle. [Online]. Available: <https://support.phishingtackle.com/hc/en-gb/articles/4410170814609-YARA-Rule-Examples>.

[10] “YARA: The Language of Malware Detection,” Medium. [Online]. Available: <https://medium.com/@heyvazov15/yara-the-language-of-malware-detection-9832496c5423>.

[11] “YARA,” Out3r Space. [Online]. Available: <https://out3r.space/2021/08/26/yara/>.

[12] “CTF Write-Ups: YARA,” Jon Jepma. [Online]. Available: <https://www.jonjepma.com/ctf-write-ups/yara/>.

[13] “YARA,” VirusTotal. [Online]. Available: <https://virustotal.github.io/yara/>.

[14] “MalwareBazaar: Threat Intelligence Repository,” ScamSandbox. [Online]. Available: <https://scamsandbox.com/blog/malwarebazaar-threat-intelligence-repository/>.

[15] “How It Works,” VirusTotal Documentation. [Online]. Available: <https://docs.virustotal.com/docs/how-it-works>.

[16] “ICS-CERT,” Kaspersky. [Online]. Available: <https://ics-cert.kaspersky.com/>.

- [17]“Velociraptor Documentation,” Velociraptor. [Online]. Available: <https://docs.velociraptor.app/>.
- [18] M. Rashid, Y. Singh, and S. I. Manzoor, “Methodology for Assessing Existing Vulnerabilities in Modbus Protocol,” *Procedia Computer Science*, 2025, doi: 10.1016/j.procs.2025.04.154. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050925012566>.
- [19] “ATT&CK Technique T1027.001,” MITRE ATT&CK. [Online]. Available: <https://attack.mitre.org/techniques/T1027/001/>.