ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Enabling Malware Analysis for IoT Devices using Remote Trusted Execution Environments

Georgios Sapounas

Thesis submitted in partial fulfillment of the requirements for the

Diploma in Electrical and Computer Engineering

Thesis Committee:

Associate Professor Sotirios Ioannidis (supervisor)
Professor Apostolos Dollas
Professor Antonios Deligiannakis

School of Electrical and Computer Engineering
Technical University of Crete

Chania, July 2023

Abstract

The proliferation of Internet of Things (IoT) devices has raised significant concerns regarding the privacy and security of sensitive data processed by these devices. In response to these challenges, this research presents a novel cloud-based malware detection solution that leverages Intel SGX enclaves, offering robust privacy-preserving guarantees for IoT devices transmitting sensitive data to remote infrastructure for malware analysis. The proposed system consists of a lightweight client application and a centralized server side infrastructure that exploits hardware-assisted encryption and remote attestation capabilities.

By offloading the computationally intensive task of malware analysis to remote servers, hosted within SGX enclaves, a secure environment is established, effectively shielding the transfer and processing of user data, even in untrusted infrastructures. This solution not only addresses the inherent security and privacy concerns of data offloading but also optimizes IoT resource utilization, providing an efficient and secure framework for malware detection in IoT environments. The research outcomes contribute to the advancement of signature-based malware detection for IoT ecosystems and serve as a blueprint for enhancing other security systems that leverage user-level enclaves in the IoT domain. Furthermore, the proposed solution ensures secure communication, attestation, and data management, thereby offering a practical and scalable approach to protect private user data from malicious entities and potential inquisitive service providers. The research findings address the critical need to safeguard sensitive data in IoT environments and provide valuable insights into preserving privacy and security in the era of interconnected devices.

Περίληψη

Η εξάπλωση των συσκευών Internet of Things (IoT) έχει εγείρει σημαντικές ανησυχίες σχετικά με την ιδιωτικότητα και την ασφάλεια των δεδομένων που επεξεργάζονται από αυτές τις συσκευές. Σε απάντηση σε αυτήν την πρόκληση, η έρευνά μας παρουσιάζει ένα καινοτόμο σύστημα εντοπισμού κακόβουλου λογισμικού, βασισμένο σε απομακρυσμένη εκτέλεση. Το σύστημα, χρησιμοποιώντας Intel SGX enclaves, προσφέρει σημαντικές εγγυήσεις για την ιδιωτικότητα των δεδομένων που μεταδίδονται στις απομακρυσμένες υποδομές για την ανίχνευση κακόβουλου λογισμικού. Το προτεινόμενο σύστημα αποτελείται από μια εφαρμογή-πελάτη και μια κεντρική υποδομή από την πλευρά του διακομιστή που εκμεταλλεύεται τις δυνατότητες κρυπτογράφησης και απομακρυσμένης βεβαίωσης με υποβοήθηση υλικού. Με την εκφόρτωση του υπολογιστικά εντατικού έργου της ανίχνευσης κακόβουλου λογισμικού σε απομακρυσμένους διακομιστές, που παρέχουν Intel SGX enclaves, δημιουργείται ένα ασφαλές περιβάλλον, το οποίο θωρακίζει αποτελεσματικά τη μεταφορά και την επεξεργασία των δεδομένων, ακόμη και σε μη αξιόπιστες υποδομές. Αυτή η λύση όχι μόνο αντιμετωπίζει τις εγγενείς ανησυχίες για την ασφάλεια και το απόρρητο της εκφόρτωσης δεδομένων, αλλά επίσης βελτιστοποιεί τη χρήση των πόρων, παρέχοντας ένα αποτελεσματικό και ασφαλές πλαίσιο για τον εντοπισμό κακόβουλου λογισμικού σε περιβάλλοντα IoT. Τα αποτελέσματα της έρευνας αυτής συμβάλλουν στην πρόοδο της ανίχνευσης κακόβουλου λογισμικού που βασίζεται σε υπογραφές για οικοσυστήματα IoT και χρησιμεύουν ως σχέδιο για την ενίσχυση άλλων συστημάτων ασφαλείας που αξιοποιούν ενσλαες σε επίπεδο χρήστη στον συγκεκριμένο τομέα. Επιπλέον, η προτεινόμενη λύση προσφέρει ασφαλή επικοινωνία και διαχείριση δεδομένων, προσφέροντας έτσι μια πρακτική και επεκτάσιμη προσέγγιση για την προστασία των ιδιωτικών δεδομένων των χρηστών από κακόβουλες οντότητες ή πιθανή παρακολούθησή τους από τους παρόχους των απομακρυσμένων υποδομών. Τα ευρήματα της έρευνας αντιμετωπίζουν την κρίσιμη ανάγκη προστασίας ευαίσθητων δεδομένων σε περιβάλλοντα IoT και παρέχουν πολύτιμες γνώσεις για τη διατήρηση του απορρήτου και της ασφάλειας στην εποχή των διασυνδεδεμένων συσκευών.

Acknowledgments

I would like to take this opportunity to express my sincere gratitude and profound appreciation to Associate Professor Sotirios Ioannidis for his invaluable mentorship as my supervisor, which has enabled me to undertake and conduct my research, expanding my knowledge in the field of hardware and software significantly.

I wish to express my deep appreciation to Dr. Dimitrios Deyannis for his unwavering guidance and steadfast support throughout the entirety of my thesis endeavor.

Furthermore, I am profoundly thankful to Professor Apostolos Dollas and Professor Antonios Deligiannakis for their esteemed roles as committee members, as well as for their meticulous assessment of my thesis.

Lastly, I extend my heartfelt thanks to my beloved family and cherished friends, whose trust and continuous support have been a constant source of motivation throughout the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the proliferation of Internet of Things (IoT) devices, ensuring the privacy and security of sensitive data processed by these devices has become a pressing concern. To address this challenge, we propose a cloud-based malware detection solution that leverages Intel SGX enclaves, offering strong privacy-preserving guarantees for IoT devices transmitting sensitive data to remote locations for malware analysis. By offloading malware analysis to remote servers, within SGX enclaves, we create a secure environment that shields the transfer and processing of user data, even in untrusted infrastructure. Our motivation stems from the need to protect private user data from malicious entities and potentially honest-but-curious service providers.

We specifically aim to overcome the performance limitations introduced by Intel SGX in signature-based analysis systems, such as intrusion or malware detection, by mitigating their impact and optimizing resource utilization. The research outcomes will not only advance signature-based malware detection for IoT devices but also serve as a blueprint for enhancing other security systems leveraging user-level enclaves in the IoT domain. Additionally, the performance improvement methodologies derived from this work will benefit multiple components of our proposed security stack, ensuring secure communication, attestation, and system call handling in enclave-based software as a service.

1.1 Motivation

The literature on malware detection often involves works that rely on complex systems that require high computational capacity or external/custom hardware. These modifications cannot always be applied on small IoT devices due to their limited hardware capabilities and software stacks. Also, maintaining full-scale antivirus solutions on every device found in a large network of interconnected IoT devices presents multiple implementation and management issues.

Moreover, the malware protection systems themselves can fall victims of malicious parties, aiming to disrupt or tamper their execution. Thus, requiring a way

to verify their correct and uninterupted execution. A possible approach to this issue could be the utilization of a Trusted Execution Environment (TEE) to safeguard the protection system's operation. The most common TEE available to IoT devices is ARM's TrustZone but developing software based on TrustZone can be an even more challenging task since it requires control of each device's firmware. This approach however is not always possible and does not scale since an IoT network may contain multiple and different devices, some of which may not be able to support the technology.

A solution to these problems is the utilization of a centralized malware detection system that provides its functionalities as a service. In this way, multiple devices can connect and offload the malware detection process, utilizing very minimal client applications that can easily be developed for a plethora of devices. Also, this approach minimizes the computation requirements for the client applications and preserves the limited resources offered by most IoT devices. However, in such a configuration, the management of the transmitted, and most possibly sensitive, device data has to be carefully carried out.

Leveraging a remote server that can be executed on more powerful X86 infrastructure provides multiple benefits. First, we can ensure that the malware detection software is always up to date, providing analysis with the latest and most up-to-date malware signature data, where rules need to be added to a single point. Also, we are able to monitor the entire network of IoT devices, perform valuable analytics and compare the results. Furthermore, we can provide multiple server instances for load balancing and dynamically change the enforced policies for malware mitigation centrally, with the server notifying the various devices and proposing the appropriate policies based on the device and identified threats.

In this work, we base our malware detection service on Intel's x86 platform, since it provides another great benefit, the utilization of the Intel SGX Trusted Execution Environment. SGX is the most versatile and extendable user-level enclave-based TEE up to this date and enables us to completely safeguard the malware detection service, its metadata as well as the transmitted client data. This technology enables the creation of protected and hardware-assisted isolated software containers in the user-space that act as reverse sandboxes. Thus, code and data enclosed in these containers, called enclaves, cannot be read or modified by any other process aside from the one utilizing them. This includes the operating system's kernel, other processes or even debugging tools. By transferring the client's data in an encrypted format and only decrypting and processing them in the server's secure enclaves we can guarantee that they are never accessible throughout the path from the IoT device to the malware detection service. Also, with this approach, sensitive data are never exposed in the server's file system or DRAM, thus being inaccessible even when transmitted to possibly tampered hosts or accessed by honest-but-curious infrastructure providers.

SGX uses hardware-assisted encryption to protect the confidentiality of the enclave pages, even when assuming an untrusted infrastructure. Enclave data can be securely sealed and exported in the untrusted infrastructure if needed, since they

now contain the required metadata that verify their integrity upon reuse. Finally, SGX provides Remote Attestation capabilities, enabling the client IoT devices to attest and verify the validity of the server at any point during the execution of the remote malware analysis.

1.2 Contributions

The contributions of this work are summarized as follows:

- We propose a practical cloud-based malware detection solution that aims to provide malware detection service to a plethora of IoT devices without exhausting their limited resources.
- Our system prioritizes strong privacy-preserving guarantees for clients with emphasis on the secure remote analysis of their sensitive data.
- We present the methodology that can be used to leverage user-level enclaves in designing a signature detection engine able to preserve its security properties even when residing in untrusted or hostile infrastructure.

1.3 Outline

The rest of this thesis is organized as follows. Chapter 2 provides an in-depth background on the most popular malware detection techniques and algorithms for malware signature identification. Also, it provides an overview of the available Trusted Execution Environments and explores prominent TEEs such as Intel SGX, Arm TrustZone, AMD SEV-SNP, AMD PSP, and Apple Secure Enclave. The chapter examines the selection process for an appropriate TEE based on specific requirements and criteria, taking into consideration factors such as security, performance, and compatibility.

Chapter 3 provides the design overview of our system's critical components, including the client, server, database, registration process, and attestation. Also, it provides our work's threat model. Then, Chapter 4 describes the implementation details of the components comprising our malware detection system. It presents essential functionalities such as establishing connections, managing whitelists, implementing encryption mechanisms, efficient data parsing techniques, and the malware detection engine.

Chapter 5 presents the evaluation of our system, focusing on the workloads, infrastructure and methodology utilized to evaluate the performance and effectiveness of our system. We provide the results obtained by analyzing the performance of each component and involved process as well as end-to-end performance metrics.

Chapter 6 provides a summary of prior research and related work in the field, highlighting the advancements, challenges, and limitations in existing approaches. Finally, Chapter 7 concludes this thesis by summarizing the key contributions, achievements, and provides directions for future work.

Chapter 2

Background

2.1 Malware Analysis

In general, malware analysis is the process of identifying and classifying malicious software, commonly known as malware, that can compromise the security and functionality of computer systems, networks, and devices. It involves the use of various techniques and technologies to detect the presence of malware and protect against its harmful effects.

Malware can take many forms, such as viruses, worms, Trojans, ransomware, spyware, and adware. Its primary objective is to gain unauthorized access, steal sensitive information, disrupt operations, or cause damage to the targeted system. Malware analysis plays a crucial role in maintaining the integrity and security of computer systems and preventing potential threats. Below, we present a set of commonly employed techniques in malware analysis:

Static file analysis Static file analysis is a fundamental technique employed in the realm of malware detection, aiming to scrutinize files without their execution. This approach involves a meticulous examination of file attributes, structure, content, and metadata to identify potential indicators of malicious activity. During static file analysis, key attributes, including file headers, extensions, import/export functions, and embedded strings, undergo rigorous scrutiny to ascertain the presence of malicious traits. By detecting patterns such as obfuscated code, encrypted payloads, or anomalous file formats, static file analysis aids in the identification of known malware and the identification of suspicious patterns. This proactive approach enables the preemptive detection and mitigation of potential threats before their execution, bolstering overall system security. However, it may have limitations in detecting sophisticated or polymorphic malware that actively tries to evade detection through code manipulation techniques.

Dynamic malware analysis Dynamic malware analysis is a robust technique employed in the examination of malware behavior in real-time, achieved by execut-

ing the malware within a controlled environment. This approach typically involves running the malware in a secure sandbox or virtual machine, facilitating meticulous monitoring of its activities to discern its functionality and potential repercussions. Throughout the process of dynamic analysis, comprehensive observations are made on the malware's interactions with the operating system, including system calls, registry modifications, network connections, and file accesses, providing valuable insights into its operational characteristics and potential impact. This approach allows for the detection of malicious behaviors that may not be apparent through static analysis alone, such as the ability to evade detection or exploit vulnerabilities. Dynamic malware analysis is an essential tool in understanding and mitigating emerging threats, enabling security researchers to gather valuable insights and develop effective countermeasures.

Dynamic monitoring of mass file operations Dynamic monitoring of mass file operations is an advanced technique that focuses on real-time observation and analysis of large-scale file modifications or transfers occurring within a system or network. This approach entails vigilant monitoring of file-related activities, including file creations, deletions, modifications, and transfers, coupled with in-depth analysis of associated metadata and behavior patterns. By dynamically monitoring these mass file operations, potentially suspicious activities can be analyzed and identified, such as an abrupt surge in file transfers or unauthorized alterations to critical files. This technique serves as a proactive measure to detect and mitigate potential data breaches, ransomware attacks, or insider threats involving the manipulation or unauthorized exfiltration of a substantial volume of files.

File extension blocklisting File extension blocklisting, also known as file type blocklisting, is a technique used in cybersecurity to restrict or block specific file types from being accessed or executed. It involves maintaining a list of file extensions that are considered potentially dangerous or prone to carrying malicious payloads. When implemented, any file with a blocked extension is prevented from being opened, downloaded, or executed within a system or network. This approach helps prevent the execution of known file types associated with malware, such as .exe, .bat, or .dll, thereby reducing the risk of infection. File extension blocklisting is a proactive measure that adds an extra layer of defense against potential threats by restricting the handling of files that are commonly used to distribute malware.

Application allowlisting Application allowlisting, also referred to as application whitelisting, is a security practice that involves creating a list of approved applications that are allowed to run within a system or network. It works on the principle of only permitting the execution of pre-approved applications while blocking all others. This approach helps prevent unauthorized or malicious software from running, as only applications on the allowlist are granted execution privileges. By maintaining a strict control over the software environment, applica-

tion allowlisting enhances security by mitigating the risks associated with unknown or unauthorized applications. It helps protect against various threats, such as malware, ransomware, and unauthorized software installations, providing a proactive defense mechanism against potential cyberattacks.

Malware honeypots A malware honeypot, also known as a honeypot file, is a decoy system or file designed to attract and trap malicious actors. It is a security mechanism that lures attackers into interacting with a simulated target, allowing researchers to gather valuable information about their tactics, techniques, and intentions. A honeypot file can be a seemingly vulnerable document, script, or executable, purposely left exposed to entice hackers. By monitoring and analyzing the activity surrounding the honeypot, security professionals gain insights into the latest malware threats, attack vectors, and exploitation techniques. Honeypots provide a valuable tool for understanding the evolving landscape of cyber threats and enhancing overall system defenses.

Checksumming Checksumming, also known as cyclic redundancy check (CRC), is a method used to verify the integrity and authenticity of data. It involves calculating a unique value, known as a checksum or CRC, from the data being transmitted or stored. The checksum acts as a fingerprint of the data, allowing comparison with the calculated value at the receiving end to detect any errors or alterations. By comparing the checksums, data corruption due to transmission errors or malicious tampering can be detected. Checksumming/CRC is widely used in various applications, including network protocols, file transfers, and data storage, to ensure data integrity and provide a reliable means of error detection.

File entropy File entropy refers to a measure of the randomness or unpredictability of the data within a file. It quantifies the information content of the file and can be used to detect changes in the file's data. The entropy value is calculated based on the frequency distribution of bytes or bits within the file. When a file undergoes changes, such as encryption, compression, or the insertion of malicious code, the entropy level tends to increase. By monitoring the entropy of files, security systems can identify suspicious or potentially harmful modifications. This helps uncover hidden or obfuscated data within files and detects deviations from normal entropy patterns.

Machine learning behavioral analysis Machine learning behavioral analysis is a technique used in cybersecurity to detect and identify malicious activities based on patterns and behaviors exhibited by software or network entities. It involves training machine learning algorithms on large datasets of normal and anomalous behaviors to recognize patterns indicative of malicious intent. By leveraging techniques such as anomaly detection, clustering, and classification, machine learning models can learn to identify and differentiate normal behaviors from suspicious or

malicious behaviors. This approach enables the detection of previously unknown or evolving threats that may not match known signatures. Thus it enhances the effectiveness of security systems by providing proactive and dynamic defense against emerging cyber threats.

Signature-based detection Signature-based detection is a common approach in malware detection that relies on comparing the patterns and signatures of known malware with the files or processes being analyzed. Malware signatures are unique identifiers derived from specific characteristics or code snippets of known malware. By maintaining a regularly updated database of signatures, antivirus software can quickly scan files and detect malware based on these predefined patterns. This method of malware detection provides an efficient means to identify and mitigate known malware threats, contributing to a proactive defense strategy against malicious activities within systems and networks.

2.1.1 Choosing the Appropriate Technique

In our proposed approach, we advocate for the utilization of signature-based detection as the optimal strategy to swiftly address the identified challenges, driven by its effectiveness and efficiency in detecting and mitigating known threats within the targeted system or network. Signature-based detection stands out among the listed techniques due to its remarkable ability to identify and neutralize known threats very fast by matching specific patterns or signatures associated with malware or viruses. The primary reason for choosing signature-based detection is its robust reliance on a comprehensive signature database, which empowers the approach to swiftly and accurately detect potential infiltrations, effectively minimizing the risk of system or network compromise. Moreover, signature-based detection distinguishes itself by its efficiency and real-time scanning capabilities, enabling proactive defense against known malicious activities. As opposed to alternative methods, such as behavioral analysis or static file analysis, signature-based detection can be achieved with low complexity and executed without special requirements for powerful hardware resources.

2.1.2 Pattern-based Malware Analysis

Pattern-based malware analysis is an essential methodology, extensively employed within the realm of cybersecurity to ascertain and dissect malicious software by leveraging pre-established patterns or signatures. This approach entails comprehensive scrutiny of distinct attributes, behavioral attributes, and code sequences affiliated with recognized malware strains, subsequently culminating in the creation of signatures aimed at detecting analogous instances in subsequent occurrences. By adopting this systematic technique, the ability to detect and mitigate potential threats posed by malicious software within various environments is enhanced. The

process of performing malware analysis through pattern matching can be delineated into the following phases:

Signature Creation Experienced security researchers and analysts conduct comprehensive examination of established malware samples to meticulously identify discrete and singular patterns that can function as discernible indicators of malicious code. These patterns encompass diverse elements, including specific character strings, byte sequences, cryptographic constants, API invocations, file or registry alterations, network communication patterns, and behavioral sequences. Through rigorous analysis, these distinctive patterns are discerned and catalogued, enabling the development of a robust signature-based detection system that effectively identifies and mitigates known threats within a targeted system or network.

Signature Database Creation The discerned patterns are systematically compiled into a comprehensive signature database, commonly referred to as a signature repository. This repository is diligently managed by antivirus vendors, security organizations, or researchers, and undergoes regular updates to incorporate newly discovered signatures in response to emerging malware variants. By ensuring the database remains up-to-date, security tools are equipped with the most current information necessary to proficiently identify and counter known malware threats, thereby bolstering their efficacy in safeguarding targeted systems or networks.

Scanning and Detection During the malware scanning process, antivirus software or other security tools systematically compare files, processes, or network activity against the meticulously stored patterns within the signature database. This comparison encompasses a range of techniques, including string matching, regular expressions, hash-based lookups, or specialized algorithms specifically designed for distinct types of signatures. Upon detecting a correlation between the observed data and a known malware signature, it unequivocally indicates the existence of malware within the system, prompting appropriate remedial action.

Alert or Quarantine Upon successfully identifying a correspondence between the observed data and a known malware signature, the security tool promptly triggers an appropriate response mechanism. This response may entail the generation of an alert, serving as a notification to both the user and system administrator regarding the detection of the malicious software. Furthermore, the security tool may enact immediate actions, such as isolating the infected files, quarantining the affected components, or executing measures to expunge the deleterious code. These decisive steps effectively curtail the potential for subsequent harm and impede the propagation of the identified malware, thereby bolstering the overall security posture of the system.

2.1.3 Pattern-matching Algorithms

Pattern matching algorithms are fundamental techniques used in various fields, including malware analysis, to identify specific patterns within a given set of data. These algorithms systematically search for occurrences of patterns, which can be strings, sequences of characters, or other structured representations, within a larger text or data stream. Some of the pattern matching algorithms extensively utilized in the realm of malware analysis are the following:

Knuth-Morris-Pratt (KMP) Algorithm The Knuth-Morris-Pratt (KMP) algorithm [1] is a highly efficient pattern matching algorithm. It achieves its efficiency through the construction and utilization of a failure function, also referred to as the failure table. The failure function enables the algorithm to intelligently skip unnecessary character comparisons during the matching process, thereby improving its overall performance. By maintaining separate pointers for the pattern and the text, and employing the failure function to determine optimal shifts in the pattern pointer, the KMP algorithm minimizes redundant comparisons and achieves linear time complexity of $O(n + m)$, where n represents the length of the text and m denotes the length of the pattern. As a result, the KMP algorithm provides a robust and efficient solution for pattern matching tasks, making it invaluable in the field of malware analysis and related domains.

Boyer-Moore Algorithm The Boyer-Moore algorithm [2] is a powerful and widely used pattern matching algorithm. It employs two heuristics, the *bad character rule* and the *good suffix rule*, to achieve efficient pattern matching. Through a preprocessing step, the algorithm constructs lookup tables, namely the bad character table and the good suffix table, which provide information for intelligent shifts and skipping irrelevant portions of the text during matching. By leveraging these rules and tables, the Boyer-Moore algorithm minimizes redundant character comparisons, resulting in an average-case time complexity of $O(n/m)$, where n is the text length and m is the pattern length. This makes the Boyer-Moore algorithm highly efficient for detecting malware signatures within large datasets.

Rabin-Karp Algorithm The Rabin-Karp algorithm [3] is a widely utilized pattern matching algorithm. It efficiently searches for multiple patterns by leveraging hashing techniques. During the preprocessing step, the algorithm calculates the hash values for the pattern and all possible windows of the same length in the input text. By comparing these hash values with the precomputed hash value of the pattern, matches are identified and further validated through character-by-character comparisons. The Rabin-Karp algorithm employs a rolling hash technique to optimize hash calculations and handles spurious hits using additional checks. With a linear time complexity of $O(n + m)$, where n represents the text length and m denotes the pattern length, the Rabin-Karp algorithm offers an efficient solution for detecting malware signatures and pattern matching tasks within large datasets.

2.1.4 Aho-Corasick

The Aho-Corasick algorithm [4] is considered as the best option for multiple pattern searching, since it matches all signatures simultaneously. This simultaneous matching can be achieved when the set of patterns is being preprocessed. In the preprocessing phase, an automaton is built, which is eventually used in the matching phase. Also, each character of the input is processed only one time during the matching phase. The Aho-Corasick algorithm has the property that, theoretically, the processing time does not explicitly depend on the number of patterns. Let $P = p1p2...pn$ be the patterns to be searched inside a text $T = t1t2...tm$ (with lengths n and m accordingly), both sequences of characters form a finite character set Σ. The complexity of the algorithm is linear in the pattern length ν, plus the length of the given text μ, plus the number of output matches. Given a set of patterns, the algorithm constructs a state machine, that matches all patterns in the text at once, one byte at a time. Each processing action of the automaton, accepts an input event. The very first action starts with the initial state, represented with zero. Each action that accepts an input event moves the current state to the next state, based exclusively on that input.

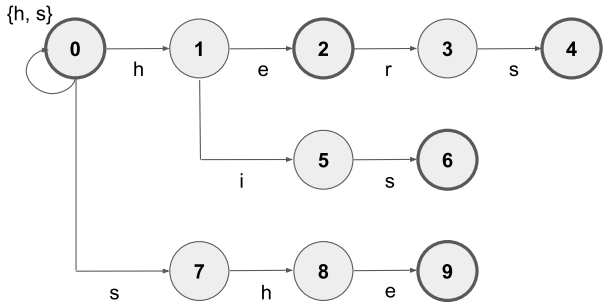
The algorithm includes three distinct functions: (i) a *goto* function, (ii) a *failure* function, and (iii) an *output* function. According to the input, one function is being triggered. Figure 4.2 present an example of these functions for the set of patterns he, she, his, hers.

The goto function, depicted in Figure **??**, plays a crucial role in determining state transitions based on the current state and the ASCII value of the input character. By examining the transitions originating from the current state, the goto function determines if a transition can be made. If the input character corresponds to one of the transitions, the next state is set to the state indicated by that transition. Conversely, if there is no matching transition, the failure function *f(i = current state)* is invoked to determine the next state. For instance, in Figure **??**, the presence of an edge labeled 'h' from state 0 to state 1 signifies that *goto(0, h) = 1*, while the absence of a transition for 'a' indicates a failure. The failure function either leads to one or more intermediate states or returns to the initial state (represented by 0 in the goto graph).

After each state transition, the output function *output(i = current state)* is checked to determine if the pattern matches a sub-string of the input text T. This process continues until the entire input text T is processed. Since failed transitions can be devoid of consuming any input, the resulting automaton is non-deterministic (NFA). Additionally, the failure function can generate multiple state transitions for a single input character. Consequently, the matching operation may require exploring multiple paths before an actual pattern match is found.

To overcome performance issues arising from large pattern sizes, an enhanced version of the traditional Aho-Corasick algorithm exists. In this revised version, all failure transitions are replaced to mitigate performance degradation. The resulting automaton, called a deterministic finite automaton (DFA), offers a single transition

per state and input character. Although this approach necessitates more memory compared to the previous one, it exhibits superior processing throughput efficiency. The complexity achieved by this approach is *O(n)*, where n represents the length of the patterns being searched.



i	output(i)
2	{he}
5	{she, he}
7	{his}
9	{hers}

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3

Figure 2.1: Aho-Corasick goto, output and failure functions.

Given the requirements of our research in malware analysis, we have chosen to utilize the Aho-Corasick algorithm due to its significant advantages over other algorithms in certain scenarios. While other algorithms have their merits and applications, the Aho-Corasick algorithm surpasses them in terms of efficiency and effectiveness. The algorithm has gained widespread recognition and adoption in renowned open-source security solutions like the ClamAV antivirus and the Snort network intrusion detection system. Its exceptional efficiency, simultaneous matching of multiple signatures, and efficient transitions between branches with shared prefixes make it a critical component in state-of-the-art malware detection systems. By leveraging the Aho-Corasick algorithm, we aim to enhance our malware analysis capabilities and achieve more accurate and efficient detection results.

2.2 Trusted Execution Environments

Trusted Execution Environments (TEEs) are specialized security technologies that play a crucial role in enhancing security within modern computing systems. TEEs create isolated environments, either in hardware or software, where critical code and data can be securely executed and protected from unauthorized access. These environments, known as secure enclaves, are shielded from external interference, ensuring the integrity and confidentiality of sensitive operations.

Table 2.1: Trusted Execution Environments.

TEE	ISA	IE	DS	RA
Intel SGX	x86_64	Y	Y	Y
ARM TrustZone	ARM	Y	Y	N
AMD SEV-SNP	x86_64	Y	N	N
AMD PSP	x86_64	Y	Y	N
Apple Secure Enclave	ARM	Y	Y	N

The technical foundations of TEEs rely on a combination of hardware and software mechanisms. Secure enclaves, implemented using hardware extensions, partition a portion of the processor's memory and execution pipeline, establishing a trusted execution environment. Additionally, TEEs incorporate components like secure boot, trusted platform modules, and attestation mechanisms. Secure boot ensures the integrity of the system's boot process, while trusted platform modules provide secure storage and cryptographic functionalities. Attestation mechanisms enable remote parties to verify the integrity of a TEE and its security properties.

TEEs find practical applications in various industries and domains. In cloud computing, TEEs enable secure execution of sensitive workloads in potentially untrusted environments, safeguarding critical data and operations. In blockchain technologies, TEEs facilitate secure and verifiable smart contract execution, ensuring the integrity and confidentiality of transactions. Moreover, TEEs are utilized in mobile devices and Internet of Things (IoT) devices to protect sensitive user data and enable secure transactions.

In our work, we choose to utilize a Trusted Execution Environment (TEE) for our pattern matching malware detection server to prioritize system security and integrity. Leveraging the TEE's hardware-backed security mechanisms, including secure enclaves and trusted platform modules, ensures robust protection for our sensitive code, data, and computations. This fortified security posture shields against attacks, unauthorized access, and manipulation attempts, bolstering the confidentiality and integrity of our algorithms.

Table 2.1 presents a compilation of widely utilized hardware-assisted Trusted Execution Environments (TEEs) available in the current landscape. The table provides information on the Instruction Set Architecture (ISA) supported by each technology, whether the TEE supports Isolated Execution (IE), secure Data Storage (DS), and Remote Attestation (RA). This comprehensive overview offers valuable insights into the key features and capabilities of these popular TEEs.

In the subsequent sections, we will explore the following widely recognized Trusted Execution Environments (TEEs): Intel SGX, ARM TrustZone, AMD SEV-SNP, AMD PSP, and Apple Secure Enclave. These TEEs have gained prominence in the field of secure computing due to their robust security features and trusted execution capabilities. We will provide detailed explanations of each TEE, highlighting their distinctive attributes and mechanisms. By examining these TEEs

individually, we aim to offer a comprehensive understanding of their functionalities and their relevance in ensuring secure and trustworthy computing environments.

2.2.1 Intel SGX

Intel Software Guard Extensions (SGX) [5] is a collection of security instructions integrated into modern Intel x86 CPUs, initially introduced with the Skylake family of processors. These instructions enable the creation of isolated software containers, known as enclaves, which provide secure and hardware-assisted execution environments. Enclaves ensure that their code and data remain inaccessible and unmodifiable by any other process, including the operating system kernel or debuggers. This isolation property significantly reduces the attack surface. SGX utilizes on-chip hardware mechanisms, such as the Memory Encryption Engine (MEE), for encryption of enclave data and code. The MEE encrypts a portion of live memory and makes it accessible to SGX, while ensuring that other processes cannot access enclave contents. Swapping of enclave memory pages to DRAM is also handled by SGX, encrypting the data to maintain confidentiality. The available memory for Intel SGX enclaves ranges from 64MB to 128MB, configurable through BIOS settings, and developers can utilize swapping techniques to access additional memory if needed. An overview of a system's attach surface with and without SGX is presented in Figure 2.2.

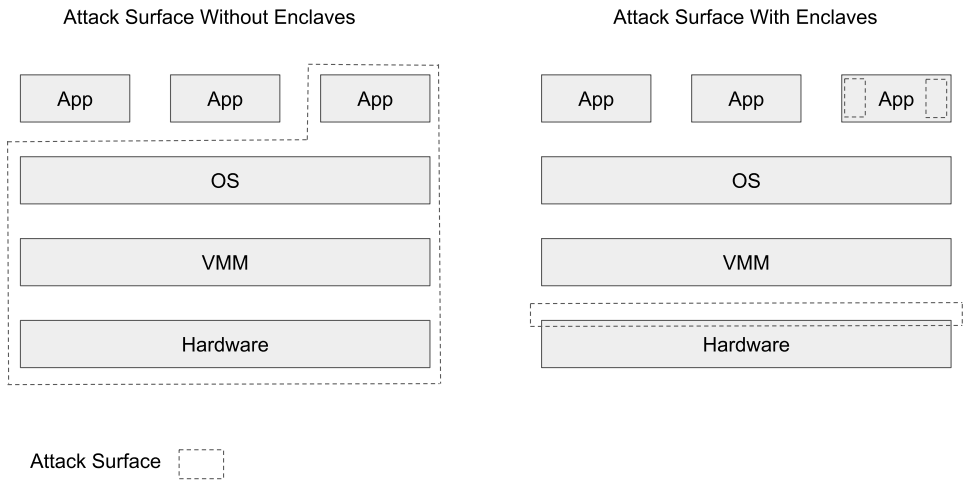


Figure 2.2: Attack-surface areas with and without Intel SGX enclaves.

Nevertheless, it is important to note that memory page swapping is exclusive to Linux systems, and Intel's SGX driver does not support this feature. As a result, the usable memory within enclaves is confined to a maximum of 128MB. On a

positive note, enclave data can be securely sealed and exported to the untrusted file system, ensuring encryption of the data in transit. To maintain data integrity, accompanying metadata is included, enabling integrity checks when the sealed data is later accessed and reused.

An SGX application typically comprises two main components. The first is the untrusted application, residing in the untrusted operating system (OS) and communicating with the secure enclave. The second is the secure enclave, which can be associated with one or multiple applications.

Communication between these components relies on specific functions and APIs declared in the SGX Enclave Definition Language (EDL) during software development. These interfaces are immutable and cannot be modified or extended after compilation and enclave signing. Enclaves are restricted from directly performing undeclared I/O, accessing system calls, or invoking privileged instructions, as the host OS kernel is untrusted and inaccessible. To handle such requests, developers must proxy them to the untrusted part of the application using designated OCALLs, which transfer execution outside of the secure enclave and can only be invoked by the enclave itself. Similarly, applications can invoke ECALLs, which transfer execution from the untrusted application to the trusted enclave at predefined entry points, triggering specific enclave functions. The execution flow of an SGX-enabled application follows a predefined pattern, as illustrated in Figure 2.3

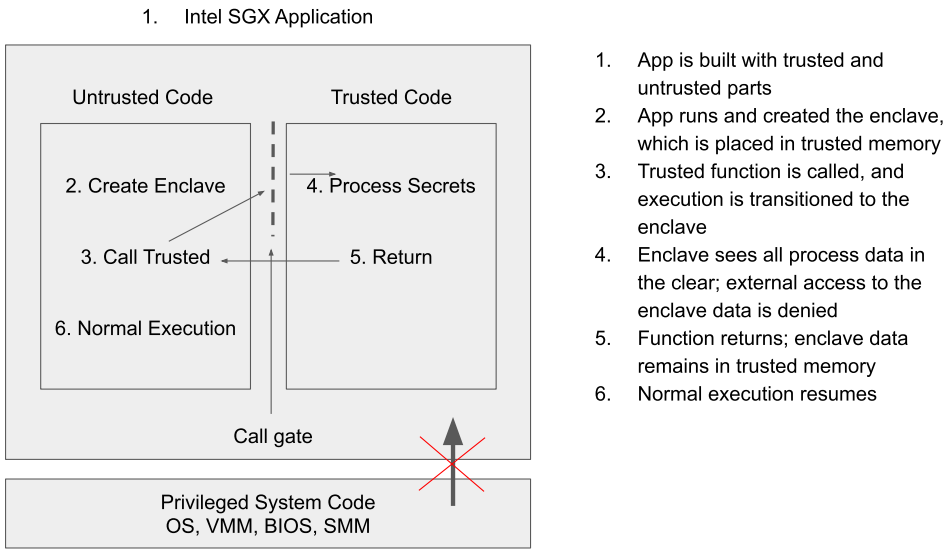


Figure 2.3: Intel SGX application execution flow.

Remote attestation is the process of verifying the authenticity of a software component running within an isolated container to a remote entity. In the context of SGX, the attested software is a secure enclave created by the trusted CPU hardware. During the remote attestation procedure, the CPU generates a measurement

for the attested enclave, providing a unique identifier. This measurement is then signed by the privileged Quoting Enclave, resulting in an attestation signature (QUOTE).

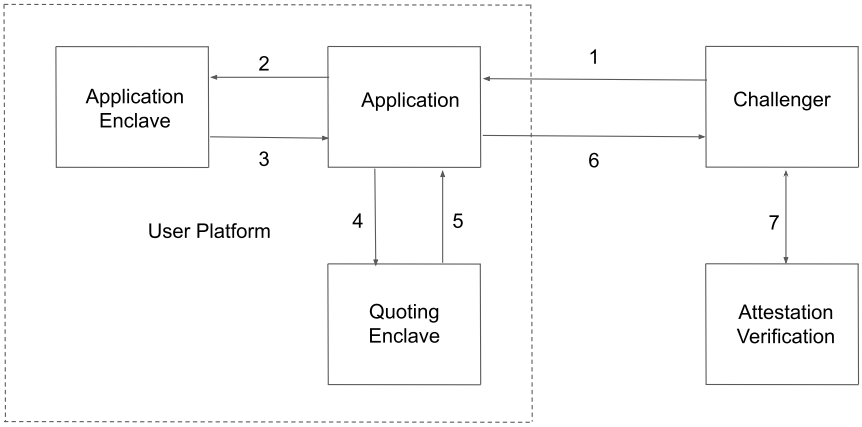


Figure 2.4: Intel SGX attestation flow.

Intel possesses the SGX hardware attestation key responsible for signing the measurement. To maintain privacy, the attestation signature employs the EPID group signature scheme. Secure communication between enclaves necessitates the establishment of a secure channel through local attestation. The attestation signature is then transmitted to a remote party, who forwards it to the Intel Attestation Service (IAS) for verification. This validation enables the remote party to determine if the enclave has been tampered with or if the attested software is executing within a genuine hardware-assisted SGX enclave. It ensures that the SGX enclave operates on SGX-enabled hardware and not in simulation mode, thereby preventing access through debugging utilities. The SGX Remote Attestation process employs a modified SIGMA protocol, enabling the remote party and the enclave to establish a shared secret for secure communication. Figure 2.4 provides an overview of the Intel SGX attestation process. Unlike Trusted Platform Modules (TPMs), SGX Remote Attestation offers improved performance as the attested software runs within the CPU. Additionally, SGX employs an EPID group signature scheme, ensuring that attested enclaves cannot be uniquely traced back to a specific CPU through their attestation signature.

Intel SGX2, the second iteration of Intel Software Guard Extensions (SGX), introduces several key features that enhance the security and functionality of the technology compared to its predecessor, Intel SGX1. One notable addition is the provision of extended memory size, enabling larger secure enclaves and facilitating

the execution of more complex and resource-intensive applications within the protected environment. Furthermore, SGX2 introduces dynamic attestation, enabling secure enclaves to establish their authenticity and integrity on-demand, allowing for flexible deployment scenarios. Additionally, SGX2 incorporates a new mode called Flexible Launch Control, which provides enhanced control over enclave execution, allowing for greater flexibility in managing enclave security policies. These advancements in Intel SGX2 further strengthen the platform's ability to safeguard sensitive data and code, empowering developers to build secure and privacy-preserving applications in a more versatile and scalable manner.

2.2.2 Arm TrustZone

Arm TrustZone [6] is a hardware-based security extension that introduces a system-wide security framework for processors based on the ARM architecture. Its objective is to establish a secure environment known as the secure world, alongside the conventional world, where the majority of system software resides. This dichotomy enables the isolation and safeguarding of sensitive code and data from potential security threats.

Central to the TrustZone design is the trusted execution environment (TEE), also referred to as the secure monitor. This critical component orchestrates the transition between the secure and normal worlds, ensuring the enforcement of security policies and the execution of trusted code. Effectively functioning as a gatekeeper, the secure monitor regulates access to the secure world, guaranteeing that only authorized software can operate within its confines.

During system boot-up, the processor initiates execution in the secure world, where the secure monitor is located. The secure monitor initializes the secure world environment and subsequently hands control over to the normal world, where the operating system and other software components operate. This transition is facilitated by secure monitor calls (SMC), establishing a mechanism for inter-realm communication.

The secure world and the normal world maintain separate memory spaces, providing robust isolation between the two domains. Sensitive code and data are securely stored in the memory allocated to the secure world, rendering them impervious to potential tampering from the normal world or other untrusted entities. Access to the resources of the secure world is rigorously controlled, effectively thwarting unauthorized access or modifications. Interactions between software running in the normal world and the secure world are facilitated by issuing SMC instructions. These instructions trigger a context switch to the secure world, enabling the secure monitor to execute requested operations on behalf of the normal world software. This mechanism empowers the execution of secure operations, including key management, secure storage, and secure communication.

TrustZone exhibits a flexible and scalable security solution, offering system designers the opportunity to define their own security policies and the partitioning parameters between the secure and normal worlds. By serving as a foundation

for constructing secure systems, it has gained widespread adoption across various devices, spanning smartphones, tablets, and Internet of Things (IoT) devices.

2.2.3 AMD SEV-SNP

AMD SEV-SNP (Secure Nested Paging) [7] is an advanced security feature designed to enhance the protection of virtualized environments. Building upon the existing AMD Secure Encrypted Virtualization (SEV) technology, SEV-SNP extends the capabilities to deliver enhanced memory security for virtual machines (VMs). The primary objective of SEV-SNP is to isolate VMs from each other and the hypervisor, effectively safeguarding them against diverse software and hardware attacks.

SEV-SNP addresses the vulnerabilities associated with speculative execution attacks, side-channel attacks, and hypervisor-related weaknesses by employing robust memory encryption techniques. It ensures that the memory of VMs is encrypted and segregated from the underlying system. Hardware-enforced memory protection mechanisms are leveraged to restrict unauthorized access to VM memory, bolstering overall security.

Key to SEV-SNP's functionality is the introduction of nested page tables, which optimize memory management in virtualized environments. Furthermore, a hardware-based root of trust is established to manage encryption keys, ensuring that only authorized VMs can access their encrypted memory regions. Moreover, SEV-SNP facilitates the seamless live migration of encrypted VMs. This capability enables the secure relocation of VMs across physical hosts while preserving encryption and the associated protective measures, thereby enhancing the flexibility and resilience of virtualized environments.

In summary, AMD SEV-SNP represents a significant advancement in virtualization security, providing enhanced memory protection and isolation for VMs, bolstered by advanced encryption, nested page tables, and secure migration capabilities.

2.2.4 AMD PSP

In contrast to Intel's Platform Trust Technology (PTT), AMD processors incorporate a distinct security mechanism called the Platform Security Processor (PSP) [8], also known as AMD Secure technology. Integrated within AMD processors since approximately 2013, the PSP operates as a trusted runtime environment with a range of responsibilities. These include managing the boot process, initializing security-related mechanisms, monitoring system behavior for potential anomalies, and implementing appropriate security responses.

The PSP can be conceptualized as an ARM kernel, with the integrated TrustZone extension functioning as a coprocessor within the primary CPU. AMD follows a firmware signing process to ensure the authenticity of the PSP firmware, which is then distributed through UEFI image files. The firmware execution occurs prior to

the main CPU boot and coincides with the loading of the fundamental UEFI. Functioning within the same system memory space as user applications, the firmware is allowed unrestricted access to Memory-Mapped I/O (MMIO).

While there are architectural and implementation disparities between AMD's PSP and Intel's PTT, both comply with the Trusted Platform Module (TPM) security protocol. This alignment ensures that both mechanisms deliver comparable security functionality, despite their underlying technical differences.

2.2.5 Apple Secure Enclave

The Apple Secure Enclave [9] is an isolated subsystem integrated into Apple's A-series chips, providing a secure execution environment for sensitive operations on devices. Operating independently from the main CPU, it has segregated memory and processing resources. Through a unique identifier and secure boot process, it establishes a chain of trust, ensuring software integrity and restricting access to authorized code for sensitive data. The Secure Enclave plays a crucial role in security features like Touch ID, Face ID, Apple Pay, and encrypted data storage.

In addition to its primary functionalities, the Secure Enclave offers a highly secure environment for executing sensitive code and managing confidential data. It employs multiple layers of security features, including hardware-based encryption, safeguards against tampering and replay attacks, and a dedicated random number generator. Notably, the Secure Enclave's inaccessibility to the main CPU and other components significantly reduces the potential attack surface, further enhancing device security.

Overall, the Apple Secure Enclave acts as a specialized and isolated enclave within the device's architecture, ensuring robust security measures for critical operations and sensitive data. Its independent nature, secure boot process, and hardware-based security features contribute to the overall protection, integrity, and user trust in Apple devices, safeguarding the confidentiality of information and enhancing overall device security.

2.2.6 Choosing the Appropriate TEE

After a thorough technical analysis, Intel Software Guard Extensions (SGX) emerges as the superior choice for our system over the other Trusted Execution Environments mentioned above. SGX offers several advantages that make it highly suitable for this task. First, SGX provides hardware-enforced memory encryption, ensuring the confidentiality of sensitive data during pattern matching operations. Additionally, SGX allows for the creation of secure enclaves, isolated regions of code execution, which protect the pattern matching algorithm and data from unauthorized access. Moreover, SGX's fine-grained attestation mechanism ensures the integrity of the enclave and enhances trustworthiness. Finally, SGX exhibits superior performance due to its optimized design and dedicated instruction set. These technical differentiators firmly establish Intel SGX as the preferred choice for our

application using commercial hardware, guaranteeing both security and efficiency in this critical application.

Chapter 3

Design

In this section, we provide an extensive account of the design and implementation of our system's architecture. The comprehensive outline of our system can be found in Figure 3.1. The system is composed of three fundamental entities: (i) the client, which facilitates the transmission of essential files to the remote server for scanning purposes, (ii) the server, which assumes responsibility for conducting malware analysis in a manner that preserves user privacy and (iii) the database, which stores the reports and statistics generated by our malware detection tool for future usage.

The entire process of malware scanning takes place within the confines of a cloud-based server, securely enclosed within Intel Software Guard eXtensions (SGX) enclaves. This encapsulation enables us to safeguard the integrity of data processing algorithms, the signature set, and most notably, the privacy of the user's sensitive information, which has to be transmitted to the remote server.

3.1 Threat Model

Providing security applications as a service has become a very popular approach due to lower cost and maintenance complexity, as well as the ability to serve multiple client platforms simultaneously. However, in the context of malware detection, the data that need to be offloaded may contain important information about the client. More specifically, a malware detection tool has privileged access to client files that may contain configuration files, e-mails, applications, sensitive files or even logged network traffic and metadata. Transferring and processing such sensitive information has to strictly comply to security and privacy preserving standards to guarantee confidentiality.

To specify the treat model for our malware detection system, we define four different entities: (i) the client, (ii) the server, (iii) the database and (iv) the infrastructure provider. The malware detection process takes place in the remote server's SGX enclaves and the server communicates with the client via a network channel. We assume that our system's client is installed and executed on the

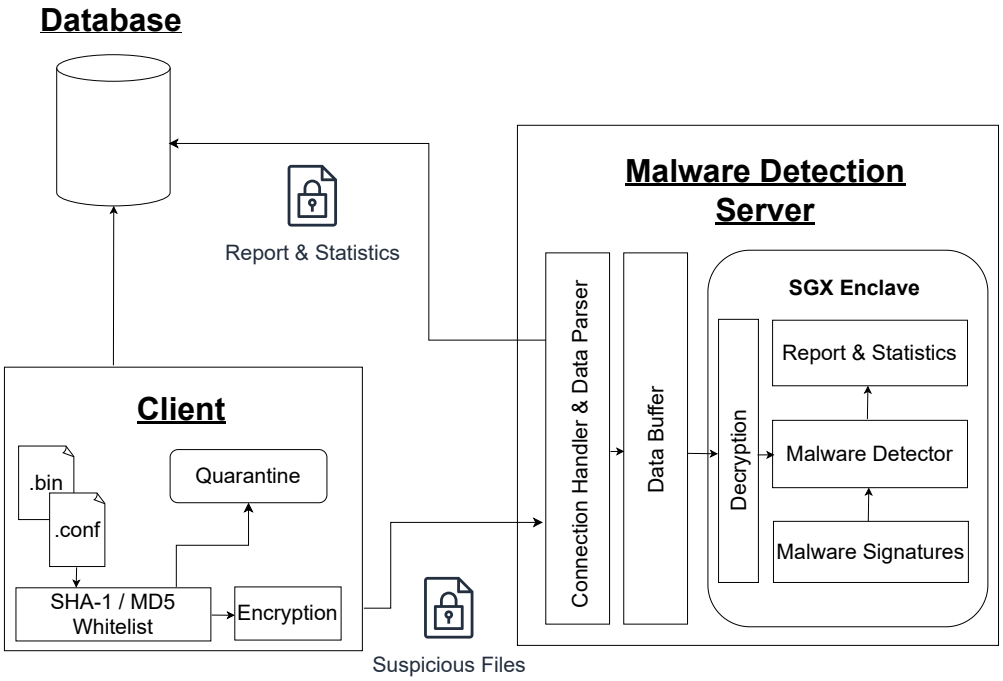


Figure 3.1: System architecture overview.

end-user device prior to any malicious party taking control of the device and its software.

The infrastructure that hosts our server is considered untrusted, since there is no control over the operating system, the possible hypervisor, the host's drivers and I/O devices, etc. Also, even if a benign environment is assumed, the possibility of an honest-but-curious cloud provider still exists.

In this work, we aim to protect both clients and server from these threats. We assume that the server and client(s) never expose their cryptographic keys to third parties. Also, we assume that the server is always executed on SGX-enabled hardware, compiled with SGX hardware mode, something that we can further attest using Intel's Remote Attestation procedure. Furthermore, we assume that the implementation of the SGX SDK, PSW, driver and required services are free of software bugs. We also exclude denial-of-service (DoS) attacks on enclaves from our threat model since the life cycle of the process handling the SGX enclaves can be controlled by a malicious operating system or superuser but without gaining any useful information by doing so. Finally, handling any side-channel attacks against Intel SGX or software flaws in SGX's implementation is out of our scope and any works that improve SGX on this direction can have a direct, drop-in, benefit to our system.

3.2 Client

The client provides three distinct functionalities, implemented through separate components. Firstly, the client prompts users to select a specific directory within the file system that they wish to scan and offers corresponding actions for any detected infected files. Second, it gathers the selected data and periodically verifies their status by comparing their hash values against a pre-defined white-list of known hash values. Last, the client transfers files with hash values that do not match the white-list to our server hosted in the cloud for malware scanning and result reporting.

The client is designed to require minimal effort and functionality, as the computationally intensive malware scanning is offloaded to our cloud-based server. This approach ensures the server's independence from the client implementation and allows for the development of multiple client versions to support different platforms and operating systems.

The hashing component within the client is responsible for retrieving data from the file system, which may include various file types such as applications, configuration files, etc. The client periodically computes the hash values of all selected files and directories, forwarding these values to the white-list component. The scanning window for periodic checks can be customized by each user based on the device type and its current status. For example, a larger window can be chosen when the system operates on battery power-saving mode to conserve device resources.

The client's white-list component compares the hash values obtained from the hash computation module against a list of hashes calculated from a known clean state of each file. If a hash value does not match its corresponding file, the file is flagged as suspicious, and the client forwards it to our server for malware scanning. Additionally, the client manages the maintenance of the white-list by handling entries for newly added or deleted files and updating existing entries with new benign hash values. Files marked as potentially infected, due to hash value mismatches, are then passed to the Data I/O module for transmission to our remote server.

The motivation behind the periodic hash-checking functionality is threefold. First, comparing hash values against a set of known hashes representing the clean state of files is a quick and efficient process, aided by the availability of various hash algorithms. Second, this approach serves as a fast preliminary filter to differentiate between benign and potentially infected files, eliminating the need for complex malware analysis on every file of a device. The aforementioned solution offers significant utility for IoT devices by effectively managing their network utilization, ensuring they do not excessively consume network resources at all times. Third, by marking only a limited subset of files as potentially infected, the amount of data to be transferred to our remote server is minimized, enhancing the overall system performance and reducing costs for users employing metered connections, mobile data plans, or low-bandwidth channels.

The secure communication component is the key element of our client, ensuring the transmission of potentially infected files to the remote server for thorough

malware analysis while maintaining security and privacy. Each file undergoes encryption using a secret cryptographic key established with the server. Following the successful transmission of marked files, the client awaits the encrypted response from the remote server. This response contains the total number of malware signatures and acts as confirmation for the client to continue. Based on the information received from the server, the client is endowed with the capability to enact file quarantine measures or maintain the existing state of the files without taking any further action. Subsequently, the white-list module updates the list for any files that have undergone benign modifications, resulting in the generation of new hash values, and removes entries for deleted files.

3.3 Server

Our server, the second component of our SGX-enabled antivirus system, is composed of three distinct modules, which are elaborated upon in the following section. The server possesses the capability to accept connections from multiple clients and conduct malware analysis on the incoming data. It can be hosted in private or public cloud infrastructures, or on a dedicated server.

One of the modules within the server is responsible for maintaining an up-to-date signature-set, which is utilized for malware analysis. By storing the entire signature-set on the remote server, the system gains two significant advantages. First, it eliminates the need for individual users to maintain the latest signature-set locally on their client devices, as the local update process may be overlooked by many users, potentially allowing undetected malware to operate on multiple devices. Additionally, users are relieved from the burden of storing the signature-set on their devices, particularly in our case of IoT devices where storage capacity is limited.

Offloading the entire malware analysis process to our cloud-based server allows us to leverage Intel SGX enclaves, even if the client device does not support them. The utilization of SGX enclaves enables the execution of the entire virus scanning process within a trusted environment. This ensures that sensitive user data, cryptographic keys, and signature-sets are never exposed in the server's Dynamic Random Access Memory (DRAM) or file system. This attribute holds critical importance for two reasons. First, it guarantees secure offloading of sensitive data to the remote server for malware analysis without the risk of data leakage. Second, even if the server is compromised, malicious actors cannot identify the signatures used in the signature-set or tamper with them. Furthermore, Intel SGX enclaves ensure secure code execution of the malware detection engine, rendering the scanning algorithms immune to attacks such as code tampering or data leakage from active variables. By operating as a reverse sandbox and establishing communication solely with the client, SGX enclaves protect user data from access even by honest-but-curious providers hosting the server, thus preserving the privacy of offloaded user data.

Upon receipt, user data are encrypted and securely transmitted to our server, where they are decrypted inside the Intel SGX enclave hosting our system's engine. The cryptographic keys necessary for successful decryption of the client's data reside exclusively within the SGX enclave. Consequently, the server's file system or DRAM never contain plain-text secret keys or sensitive data such as system configuration, binary or log files, thereby ensuring their inaccessibility to the server's host or provider. Even in the event of a compromise of the non-SGX part of our server or the hosting infrastructure, the keys, malware signatures, and private user data remain inaccessible.

The focal point of our system revolves around the malware scanning module, which operates exclusively within the secure Intel SGX enclave. Within this enclave, the module conducts thorough analysis of incoming data using a predefined set of malware signatures. Each signature rule in the set possesses a unique identifier and comprises patterns and metadata that elucidate the malware's functionalities, risk levels, and recommended actions. Utilizing the client's secret key, the virus scanning process unfolds within the enclave subsequent to decrypting the data. Once an input file triggers a rule successfully, the associated metadata and file details undergo processing and are subsequently transmitted to our database as a comprehensive status report.

Furthermore, an additional component housed within the secure enclave generates the scanning report. Our server receives the outputs from the malware scanning engine and assembles a comprehensive report for subsequent processing by our client. This report encompasses pertinent information regarding identified malicious files, including filenames, action needed and scanning timestamps.

3.4 Database

Our database serves as a centralized repository for securely storing critical data and statistics generated by our SGX-enabled antivirus system. It efficiently captures and retains essential information, including file paths, file hashes, iteration statistics, and processing times of clients, servers, and the database itself.

The database effectively captures and organizes file paths, providing a structured representation of the specific locations within the file system where scanned files are located. This structured organization enables efficient file retrieval, management, and subsequent operations or investigations.

Furthermore, the database preserves the iteration statistics of file hashes, encompassing the computed hash values of files and directories, along with relevant statistical data derived from the hashing process. These statistics capture crucial metrics such as hashing algorithm performance, processing speeds, and resource utilization. By storing and tracking these statistics, the database facilitates performance analysis and monitoring of the hashing operations, contributing to the overall efficiency of the system.

Additionally, the database records and maintains processing times for various

components within the system, including the client, server, and the database itself. This includes the time durations required for specific operations to complete by each component. By capturing and storing these processing times, the database enables comprehensive analysis of system performance, identification of potential bottlenecks, and opportunities for optimization.

In a production deployment, the ideal choice for implementing the database in our SGX-enabled antivirus system would be Microsoft SGX enabled SQL [10]. This solution combines the enhanced security benefits provided by Intel Software Guard Extensions (SGX) with the robust features and scalability of a SQL-based database management system. However, as our project is currently in the prototype phase, we have chosen to utilize free and open-source software to facilitate the development process without incurring additional costs.

3.5 Registration

The registration process serves as the initial task carried out by our system upon the initiation of our client on a user's device. In the first step of this process, the client establishes communication with our cloud-based server and performs a key exchange to establish a shared key. During this interaction, the server generates a unique client ID and securely stores the shared key, along with the corresponding ID, within the SGX enclave.

The second step involves the generation of a comprehensive list containing the hash values of each file in a clean and uninfected state. To accomplish this, our client applies a hashing algorithm to every selected file, temporarily populating the white-list with the resulting hash values. Subsequently, each file is transmitted to our remote server for thorough malware analysis.

Upon receiving the server's response in the form of a report, the hash values corresponding to the uninfected files are considered permanent entries in the white-list. Any identified malicious files, if present, are presented to the user along with suggested actions from the report. Following the completion of the registration process, our client prompts the user to specify a periodic hashing interval, with automated periodic scanning being the default setting.

3.6 Attestation

Our system can enhance its security and establish a higher level of trust in the SGX-enabled server by leveraging Intel's Remote Attestation services. Through remote attestation, our client can challenge the server to verify that the core component of the engine resides within a signed SGX enclave and is executed on an SGX-enabled processor in a trusted hardware mode. This approach effectively mitigates the risk of malicious entities masquerading as an SGX-enabled server in order to gain unauthorized access to users' private data. Additionally, it prevents entities from running our server in SGX debug or simulation mode, thereby thwarting

attempts to gain access to sensitive user data and the server's secret keys through the use of debuggers.

Chapter 4

Implementation

In this section, we describe in detail the implementation of our signature-based malware detection system, providing in depth details about the operations performed by the client and the server components. Also, we present the process of secure communication establishment and the operation of our custom version of the Aho-Corasick pattern matching algorithm, utilized for malware detection. Furthermore, we describe the operation of the SGX enclaves, the process of result and metrics logging and provide a step-by-step execution life cycle.

4.1 Client

The client is designed to address the need for secure file checking and communication in a networked environment. By implementing formal and technical methods, the client ensures reliable and tamper-proof data exchange. The client's functionalities, cover various aspects such as configuration parsing, socket creation, key exchange, initialization, file searching, and file processing.

Configuration Prasing The client exhibits a robust capability to adapt to diverse environments by leveraging configuration parsing mechanism. This enables the retrieval of critical parameters essential for customization. Parameters encompass the hash mode, such as MD5 or SHA1, the bit mode with options like 128 or 256, the interval duration between file checks, the option to search files again, the database reset functionality, and the ability to quarantine files. The client gains the flexibility to adjust its behavior based on the specific requirements of different environments.

Socket Creation and Key Exchange To establish secure and reliable communication, the client takes a proactive step by creating a TCP socket. This socket acts as a conduit for seamless data exchange between the client and the server. Moreover, the client implements a robust key exchange protocol to ensure the confidentiality and integrity of transmitted data. By engaging in this protocol, the

client verifies the authenticity of the communication parties, preventing unauthorized access or modification of the exchanged information.

Initialization and File Searching The client demonstrates meticulous preparation by initializing key components necessary for cryptographic operations. It sets up the Initialization Vector (IV) and TAG, crucial elements for maintaining data security during cryptographic operations. Leveraging the `dirent.h` library, the client performs an extensive file search within the specified directory and its subdirectories. This exhaustive scanning capability allows the client to identify all relevant files for subsequent processing.

File Processing and Communication With each discovered file, the client initiates a series of various operations. It captures and stores the file's path, hash, and ID values in a structured format, enabling efficient retrieval and reference in the future. By determining the file's length and content, the client gains a comprehensive understanding of its characteristics. Employing the configured hash mode and bit mode, the client computes an appropriate hash value. It compares this hash value to the previous check, ensuring the detection of any modifications. Employing cryptographic algorithms, the client encrypts the file's data, reinforcing its security. The encrypted data is then promptly dispatched to the server for detailed analysis. Concurrently, the client logs relevant file details, including the transmitted files, to the database. Upon server response, the client determines the file's status based on the server's evaluation.

Logging and Performance Monitoring To ensure meticulous tracking and performance analysis, the client captures and logs the precise execution times for each operation performed. This comprehensive logging mechanism allows for a thorough evaluation of the client's efficiency, effectiveness, and overall performance. By leveraging this data, the client gains valuable insights, identifies potential bottlenecks, and allows us to determine where to implement future optimizations.

4.1.1 Connection

To establish a secure and reliable communication channel with the malware analysis server, the client first creates a Transmission Control Protocol (TCP) socket. This socket acts as a conduit for seamless data exchange between the client and our server, ensuring the integrity and ordered delivery of the transmitted data. By leveraging the TCP socket, the client establishes a stable and dependable connection with our server.

The utilization of Intel SGX technology provides a secure enclave where the sensitive data for malware analysis is processed. This hardware-based security mechanism ensures that the data remains protected inside the enclave, safeguarding it from external threats. As a result, the need for additional encryption protocols such as SSL/TLS is obviated. The data transmitted between the client and

the Intel SGX server remains encapsulated within the server's secure enclave, minimizing the risk of unauthorized access or tampering during transmission. The combination of Intel SGX's trusted execution environment and the secure communication established through the Diffie-Hellman key exchange protocol ensures that the data is securely transmitted and analyzed without relying on SSL/TLS encryption. Moreover, the client implements a robust key exchange protocol based on the widely-used Diffie-Hellman algorithm, enabling secure key establishment over an insecure network without prior cryptographic material exchange. This protocol ensures the confidentiality and integrity of the transmitted data and verifies the authenticity of the communication parties, preventing unauthorized access or modification of the exchanged information.

The Diffie-Hellman [11] key exchange process involves several steps. First, both the client and our server agree upon a prime number (p) and a base value (g). These values are known to both parties. Next, each party independently selects a random secret value (a for the client and b for the server). These secret values are kept private and not exchanged over the network.

Using modular arithmetic operations, each party computes their respective public keys. The client computes its public key (A) by raising the base value (g) to the power of its secret value (a) modulo the prime number (p). Similarly, our server computes its public key (B) by raising the base value (g) to the power of its secret value (b) modulo the prime number (p). The client and the server then exchange their computed public keys.

Upon receiving the public key from the other party, each participant uses their own secret value to compute a shared secret key. The client computes the shared secret key (K) by raising the received public key (B) to the power of its secret value (a) modulo the prime number (p). Likewise, our server computes the shared secret key (K) by raising the received public key (A) to the power of its secret value (b) modulo the prime number (p).

Both the client and our server now have the required information to derive a shared secret key (K') that is known only to them. This shared secret key can be used for subsequent symmetric encryption and decryption operations, ensuring the confidentiality and integrity of the files sent during their transmission and storage. Since the shared key resides in the enclave and the client data decryption is exclusively performed within the enclave, we can ensure that sensitive client data are never exposed in plaintext format in the server's DRAM or file system; something that could not be achieved with traditional secure communication protocols.

In conclusion, by combining the utilization of a TCP socket for seamless data exchange, the implementation of a robust key exchange protocol based on the Diffie-Hellman algorithm, and the utilization of Intel SGX technology for hardware-based security, the client establishes a secure and reliable connection with our server for malware analysis. This approach ensures the confidentiality, integrity, and authenticity of the exchanged information, enabling a trusted environment for malware analysis.

4.1.2 Whitelist

The client implements a robust hash whitelist mechanism utilizing either the SHA-1 or MD5 cryptographic hash algorithm for file integrity verification. In the initial iteration, the client diligently calculates the hashes of all files present in the designated system or directory using the selected hash function. These computed hash values serve as the fundamental reference for subsequent iterations.

In each subsequent iteration, the client evaluates the current hash of each file against its corresponding previously calculated hash. This comparison enables the client to discern any alterations or unauthorized modifications that may have transpired since the inception of the hash whitelist. The successful matching of the current hash with the previous hash indicates the absence of tampering or modifications, thereby affirming the file's integrity. Conversely, a mismatch between the current and previous hashes signifies potential tampering or unauthorized alterations, compelling the client to undertake further investigative measures.

By leveraging SHA-1 or MD5, the client ensures an efficient and reliable file integrity verification process. The comparison between the current and previous hashes facilitates the identification of discrepancies without the necessity of storing or analyzing the entire file contents. This approach is particularly advantageous when dealing with extensive files or a substantial volume of files that necessitate frequent verification.

The implementation of the hash whitelist mechanism by the client significantly enhances the overall security of the system. Through the continuous monitoring and evaluation of file hashes, the client promptly detects and responds to potential security breaches or unauthorized modifications. This proactive approach safeguards the integrity and trustworthiness of the data within the system and minimizes the need for unnecessary data transfers and malware analysis.

4.1.3 Encryption

Encryption is implemented using a key establishment protocol, as described earlier. This protocol ensures secure and confidential communication between the client and the server. While the Intel SGX server leverages its hardware-based security to safeguard the data within its enclave, the client, which lacks SGX capabilities, performs encryption within its main memory. This approach ensures that sensitive information remains protected during transmission and data cannot be decrypted anywhere in the path other than in the server's enclave which holds the key.

The encryption algorithm employed is AES-128-GCM, chosen due to its robustness and availability in the client's environment. As the Intel SGX enclave does not support AES-256-GCM, AES-128-GCM provides a suitable alternative with a higher level of encryption strength. AES-256-GCM combines the Advanced Encryption Standard (AES) symmetric encryption algorithm with the Galois/Counter Mode (GCM) for authentication and integrity checks. This combination ensures both confidentiality and integrity of the encrypted data, providing a strong defense

against unauthorized access and tampering.

By utilizing the key establishment protocol for secure communication and implementing encryption with AES-256-GCM, the system ensures that sensitive information is effectively protected. The encryption process, performed within the client's main memory, adds an extra layer of security to prevent unauthorized access and maintain the confidentiality of the data. This comprehensive approach to encryption contributes to the overall security posture of the system, safeguarding against potential threats and ensuring the integrity and privacy of the transmitted and stored information.

4.1.4 Data Transmission

To facilitate focused analysis and optimize resource utilization within the SGX-enabled antivirus system, the file sending process follows a specific sequence. First, the client sends the file size in bytes, enabling the server to anticipate the total number of bytes to receive. This information allows for efficient resource allocation on the server side. Subsequently, the client transmits the actual file content, ensuring its integrity and confidentiality through encryption.

Upon the discovery of a file, the client prepares it for transmission. It captures and stores essential file attributes, such as the file's path, hash, and ID values, in a structured format. These attributes enable efficient retrieval and reference in the future, streamlining file management within the system. Additionally, the client determines the file's length and content to gain a comprehensive understanding of its characteristics, enabling subsequent analysis and processing to be tailored accordingly.

To ensure data integrity, the client computes an appropriate hash value for the file using the configured hash mode and bit mode. This hash value acts as a unique identifier and enables the client to detect any modifications or tampering that may have occurred since the last check. By comparing the computed hash value to the previous check, the client ensures the file's integrity throughout the transmission process.

To reinforce the security of the file during transmission, the client employs robust cryptographic algorithms to encrypt the file's data. This encryption process safeguards the confidentiality and integrity of the file, preventing unauthorized access or modifications during transit. By encrypting the file data, the client enhances its protection and ensures that only the server, equipped with the necessary decryption capabilities, can access and analyze its contents within its secure enclave.

During the file transmission process, the client diligently logs relevant file details, including the transmitted files, in the database. This comprehensive logging provides an auditable record of the file sending process, facilitating subsequent analysis, investigation, and reference. The database serves as a centralized repository of file-related information, contributing to efficient file management within the system.

Once all files have been sent, the client concludes the transmission by sending a special marker, represented by the string `"##EOS##"` (End of Stream), to the server. This marker signifies the end of the file stream and allows the server to process the transmitted files accordingly. It serves as a signal that no further files are expected, enabling the server to allocate its resources appropriately for analysis and evaluation.

By adhering to the sequence of sending the file size first, followed by the actual file content, and employing the `"##EOS##"` marker to signify the end of the file stream, the client ensures optimal resource utilization, and effective communication with the server during the file transmission process without exhausting the resources of the SGX-enabled antivirus system.

4.1.5 Database Updates

On the client's side, the process of updating the database involves the logging of critical statistics and processing times related to files and client operations to the central repository. During the database update, the client ensures that the repository reflects accurate file statistics, including the unique file ID, file path, and file hash. The file ID serves as a distinctive identifier within the system, facilitating subsequent operations and investigations. The file path denotes the precise location of the scanned file within the file system, enabling efficient file organization and management. The file hash, derived from the file's contents, acts as a unique identifier and enables fast file comparisons. By incorporating these updated statistics into the database, the client guarantees the repository's alignment with the current file information, even after service disruptions.

Furthermore, the client captures and updates the database with processing times associated with client operations. This includes the encryption time, representing the time required for the client to encrypt the file prior to transmitting it to the SGX-enabled antivirus system. Additionally, the client records the send time, which signifies the duration of transmitting the encrypted file to the server or database. These processing times contribute to the assessment of encryption efficiency and data transmission, facilitating analysis and potential optimizations.

Moreover, the client records and updates the database with database query times. This metric represents the duration it takes for the client to retrieve information from the central database, such as searching for specific files or retrieving relevant statistics. By monitoring query times and updating the database accordingly, the system gains insights into the performance of database operations and identifies opportunities for query optimization. By consolidating the updates for file statistics, encryption and transmission times, and database query duration, the client ensures the central repository remains auditable.

4.1.6 Quarantine & Mitigation

File quarantine is an essential component in our malware detection workflow, ensuring the safe handling and containment of potentially malicious files. When a file is quarantined, multiple actions are taken to mitigate any risks associated with the file. First, the file is permanently deleted from its original location to prevent accidental execution or inadvertent access. This proactive measure minimizes the potential harm that the file may pose to the system or network.

In addition to deletion, the quarantined file is marked as non-executable. This further reduces the risk of any accidental execution, preventing malware from executing and potentially causing harm to the system or compromising its security. By preventing the execution of quarantined files, the system ensures that the client's environment remains controlled and secure.

To preserve the file for subsequent analysis, the system employs file compression techniques. Compressing the quarantined file reduces its size, making it more manageable for storage and transmission purposes. This allows for efficient archival and future examination without consuming excessive storage resources while also prevents unwanted or accidental malware execution.

To enhance the security of the quarantined file, the client goes a step further by applying both file compression and encryption. By combining these two techniques, the file is not only compressed for efficient storage but also encrypted to prevent accidental or unauthorized access. This ensures that the malicious file cannot be decompressed and accessed either on purpose, by a malicious party, or accidentally, by the client.

The file quarantine process, involving deletion, marking as non-executable, and applying file compression with optional encryption, strengthens the system's overall security and enables the safe handling and analysis of potentially malicious files. These measures ensure that the quarantined files are securely contained, minimizing the risk of any accidental execution or unauthorized access, while maintaining the necessary data for further investigation and analysis.

4.2 Server

The server plays a vital role in secure pattern matching and database management. By employing several techniques, the server ensures the integrity and confidentiality of processed data. The server's operations include enclave initialization, data buffer initialization, DFA pattern loading, TCP socket setup, data reading, pattern matching, report generation and database updates.

Enclave and Data Buffer Initialization The server undertakes the initialization of the Intel SGX enclave. By creating a secure execution environment, the enclave safeguards the confidentiality and integrity of the critical malware detection code and the required data. Furthermore, data buffers are initialized within the enclave to optimize data handling and processing efficiency.

DFA Loading To identify specific patterns within the incoming data, the server loads the DFA generated by the available malware patterns into its enclave memory. The DFA serves as the rule-set to be used for the malware detection process. Leveraging a serialized DFA approach, as we explain further in this chapter, the server achieves a high-performance during the malware detection process.

TCP Socket Setup and Data Handling Facilitating reliable and secure data exchange with clients, the server establishes separate TCP sockets. The TCP socket configuration ensures seamless and trustworthy communication. The server reads data from each socket, employing a repeated reading process until the complete data stream is received, based on the information received upon client connection.

Pattern Matching and Database Updates Following the complete reception of data, the server engages the pattern matching function. This function exploits the preloaded DFA to swiftly identify matches within the data stream using our custom version of the Aho-Corasick pattern matching algorithm. When a match is successfully detected, the server updates the corresponding fields within the database. By doing so, the server maintains a comprehensive record of identified malware patterns and their associated data.

Logging To meticulously monitor server's performance and track its operational processes, the server logs the precise execution times of each operation within the database. This meticulous logging mechanism serves as a vital tool for measuring the server's efficiency, identifying potential bottlenecks, and pinpointing areas that may require optimization.

4.2.1 Data Handling

The server's data parsing process within the SGX enclave is critical for receiving and processing files efficiently. It involves the utilization of a function called `databuf_add()`, which is responsible for receiving and storing the specified amount of bytes for each file in the data buffer. The function is designed to manage the buffer effectively, ensuring it can handle incoming files and prepare them for analysis. When the server receives a file, the `databuf_add()` function is invoked to receive the specified number of bytes and store them in the data buffer. By incrementally adding the received bytes to the buffer, the function ensures that the file data is correctly accumulated in the intended order. It carefully tracks the buffer's size and determines when it reaches its capacity, indicating that it can no longer accommodate additional file data.

Upon reaching the buffer's capacity, the function initiates the processing of the stored data by first transferring them within the secure enclave. This processing phase involves executing various algorithms to achieve malware detection. The server performs tasks such data decryption within the enclave and threat detection

and evaluation on the received files. This analysis aims to identify potential threats, verify the files' integrity, and extract essential information for subsequent actions.

After completing the processing of the data buffer, the server prepares to receive new files by either clearing the buffer or allocating a new buffer instance. This ensures that subsequent files can be received and stored without interference or data corruption from previously processed files. The function then resumes its operation, repeatedly receiving the specified amount of bytes for each file and following the process described above. This cycle continues as long as there are incoming files to be processed, enabling the server to handle files in a continuous and efficient manner, optimizing resource utilization within the SGX-enclave.

Once the server receives the special marker `"##EOS##"` (End of Stream) from the client, it signifies the completion of the file transmission process. At this point, the server proceeds to perform additional tasks, including the evaluation of statistics and updating the relevant database fields.

Upon receiving the `"##EOS##"` marker, the server extracts the necessary information from the data buffer and initiates statistical analysis. This analysis involves aggregating and processing the data received for each file, allowing the server to calculate various statistics such as the number of files processed, average file size, processing times, and other performance indicators. These statistics provide valuable insights into the system's performance, aiding in performance analysis, optimization efforts, and overall system monitoring. In addition to generating statistics, the server updates the relevant database values based on the processed files. It logs the processed files' attributes, analysis results, and other pertinent details into the database. This comprehensive and up-to-date database serves as a centralized repository for efficient file management, historical analysis, system auditing, and supporting the decision-making processes.

By combining the functionality of the `databuf_add()` function to handle incoming file data and the processing of the `"##EOS##"` marker for generating statistics and updating the database values, the server ensures a robust and efficient data handling process within the SGX enclave.

4.2.2 Malware Analysis

Our malware scanning engine utilizes our custom version of the highly efficient Aho-Corasick pattern matching algorithm. Variations of this algorithm are widely adopted by various signature-based solutions, including the renowned open-source antivirus ClamAV. Our system ensures maximum data privacy, code security, and signature-set integrity by executing the entire malware scanning process within Intel SGX enclaves. By leveraging this secure execution environment, our system safeguards the confidentiality of offloaded data, protects the integrity of executed code, and maintains the integrity of the signature-set.

As explained earlier, traditional implementations of the Aho-Corasick pattern matching algorithm utilize a state machine structure represented as a tree. However, this structure is not optimized for use within SGX enclaves due to its memory

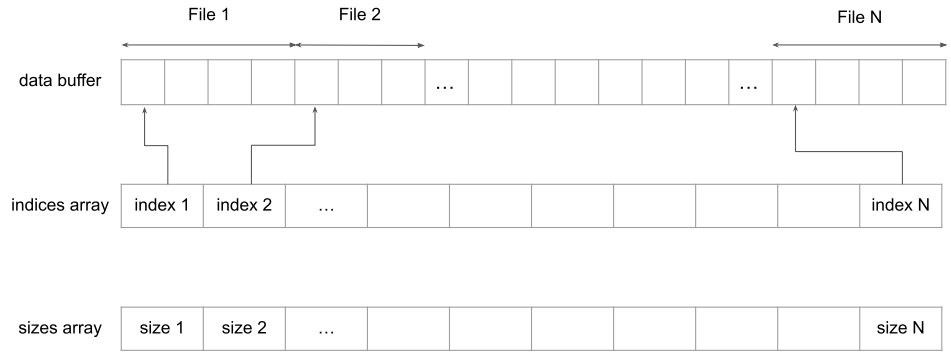


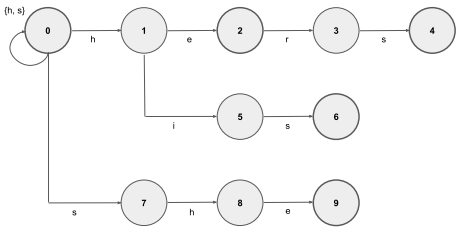
Figure 4.1: Data buffer overview.

constraints and the detrimental effect on performance caused by scattered memory access during pattern matching process.

To overcome these limitations and enhance performance, we adopt an alternative approach. We represent the DFA as a serialized version of the state machine tree, using a one-dimensional integer array. The serialization process involves converting the tree into an array which we will describe as two-dimensional array for presentation purposes. In reality, the rows of this two-dimensional array are concatenated into one. The 2D array has 256 columns to accommodate the ASCII set, representing all possible values of a single input byte. Each row corresponds to a DFA state, and each cell contains the ID of the next valid transition for the input byte it represents.

During the matching process, traversal of the serialized DFA tree starts from the initial state (row 0). The appropriate column is selected based on the ASCII value of the first input character. The next valid state is retrieved from the corresponding cell, pointing to another row. The process continues by fetching the next character from the input and moving to the cell indicated by the row and column. Negative values in the array indicate final states and signify a successful match. The search then proceeds using the absolute value as a reference for the next step. Fail states direct the matcher to a previous valid state or the initial state 0.

The serialized DFA array simplifies the traversal process, minimizes the memory requirements to store the state machine and provides significant performance benefits by enabling caching effects. Also, it enables fast and efficient malware detection with a minimal codebase since DFA traversal is performed with simple table indexing. Moreover, it reduces the Trusted Computing Base (TCB) and facil-



(a) Aho-Corasick tree.

ASCII set

States	0	...	e 101	...	h 104	i 105	...	r 114	s 115	...	255
0	0	0	0	0	1	0	0	0	7	0	0
1	0	0	-2	0	1	5	0	0	7	0	0
2	0	0	0	0	1	0	0	3	7	0	0
3	0	0	0	0	1	0	0	0	-4	0	0
4	0	0	0	0	8	0	0	0	7	0	0
5	0	0	0	0	1	0	0	0	-6	0	0
6	0	0	0	0	8	0	0	0	7	0	0
7	0	0	0	0	8	0	0	0	7	0	0
8	0	0	-9	0	1	5	0	0	7	0	0
9	0	0	0	0	1	0	0	3	7	0	0

(b) Aho-Corasick DFA serialized as an integer array.

Figure 4.2: The Aho-Corasick Deterministic Finite Automaton (DFA) is represented as an integer array and consists of patterns such as "he," "she," "his," and "hers." In sub-figure (a), the DFA is visualized with dark blue nodes indicating the final states, which mark the end of a pattern. Alternatively, in sub-figure (b), the same information is represented using negative values.

itates easy auditing to identify and eliminate any potential security vulnerabilities or software bugs.

4.2.3 Enclave I/O

Our implementation focuses on ensuring both security and efficiency during the I/O operations with the SGX enclaves, where the malware scanning code, signature-set (DFA), and user data are securely stored. The enclave only provides a single entry point for user-data input. However, due to the limitations of SGX enclaves in accessing system calls, the non-SGX enabled portion of our server manages the network sockets required for receiving client data. The data is encrypted when transmitted over the network, and the secret keys necessary for decryption are exclusively stored within the SGX enclave.

To minimize performance overhead, we employ an approach where incoming encrypted client data is batched using a buffer in the non-SGX enabled part of the application, as previously described. Once the buffer is full, it is transferred into the enclave. The size of the buffer can be dynamically optimized based on the current workload. Once a batch of user data is gathered, it is sent to the enclave for processing via an Ecall function.

Before the malware detection can be performed, the matcher within the secure enclave decrypts the user data using the appropriate key. This ensures that even if the server is compromised and blocks data forwarding into the enclave, the data, DFA, and secret keys remain protected and are not stored in plain text format in main memory or the file system. The results of the malware analysis are compiled into a report that it is sent to the database so it can be parsed by the client. This report contains information about infected files, identified malware, and recommended actions for mitigating each threat.

4.2.4 Database Updates

The server's database update process includes the tasks of updating the central repository with crucial information and precise processing times pertaining to file matches and various server-side operations. The server is responsible for updating the database with malware pattern matches corresponding to each file. These matches represent the outcomes of the file analysis conducted by the SGX-enabled malware detection system. By associating files with their respective matches, the database serves as a comprehensive repository of detected threats, vulnerabilities, or other pertinent information. This facilitates efficient retrieval and analysis of match data during subsequent operations or investigations.

In addition to file matches, the server captures and updates the database with processing times related to specific server-side operations. This includes the data receiving time, which denotes the duration required for the server to receive the encrypted file(s) from the client. The decryption time signifies the duration necessary for the server to decrypt the received file(s), ensuring its accessibility for analysis. Furthermore, the matcher processing time quantifies the time taken by the server to perform the actual analysis and match detection on the decrypted file. By incorporating these processing times into the database, the system gains insights into the performance of server-side operations, enabling the identification of potential areas for optimization and performance enhancement.

Moreover, the server records and updates the database with server database querying times. This metric denotes the duration it takes for the server to query the central database when retrieving relevant information during the analysis process. By monitoring and updating these query times in the database, the system can evaluate the efficiency of server database operations, leading to analysis-driven optimization opportunities.

By diligently updating the database with file matches, processing times (such as receive time, decryption time, and matcher processing time), and server database querying times, the server ensures the central repository accurately reflects the outcomes of file analysis and provides performance-related insights. Consequently, this enables efficient retrieval of all required data, meticulous analysis of server-side operations, and optimization of database querying within the SGX-enabled antivirus system.

4.3 Reports & Statistics

The reports and statistics captured offer valuable insights into the behavior and effectiveness of the SGX-enabled antivirus system. The metrics related to buffer entries and entry size provide an understanding of the system's capacity to handle incoming data and the granularity with which it can process individual items. The DFA memlock status reveals whether the DFA component is securely locked in memory, ensuring its integrity and protection from unauthorized access.

The metrics concerning states, cached states, and caching efficiency shed light

on the system's utilization of deterministic finite automaton technology for threat analysis. By monitoring the number of states and the effectiveness of caching, administrators can assess the system's efficiency in pattern matching and threat detection. Additionally, these metrics serve as indicators for potential performance bottlenecks or areas for improvement in the analysis process.

The metrics related to DFA size, DFA cache size, cache hits, cache misses, cache references, hit rate, and miss rate provide an in-depth understanding of the system's memory consumption, cache utilization, and cache performance. A well-optimized DFA cache with a high hit rate can significantly enhance the speed and efficiency of the system's operations, reducing the reliance on external resources and improving overall performance.

In addition to the aforementioned metrics, the statistics on processed data and throughput offer valuable insights into the system's data handling capabilities. The volume measurements in various units provide a comprehensive overview of the amount of data processed by the system. This information enables administrators to gauge the system's capacity and scalability, ensuring it can handle large data volumes without compromising performance or stability. Furthermore, the throughput metric, measured in Mbps or Gbps, provides a quantitative assessment of the system's processing speed. A higher throughput indicates a more efficient handling of incoming data streams, reducing processing delays and enhancing the system's responsiveness.

The generated reports are stored in a designated logfile within the server. This logfile serves as a persistent record of the system's performance and analysis-related information. By storing the reports in a logfile, the system ensures the availability of historical data for future analysis, comparison, and reference purposes. It also enables system administrators to track and monitor the system's performance over time, facilitating the identification of long-term trends and patterns. The logfile serves as a valuable resource for ongoing system evaluation, troubleshooting, and optimization efforts, providing a centralized repository of crucial performance-related information within the SGX-enabled antivirus system.

4.4 Execution Life Cycle

In this section, we present the complete execution life cycle of our system. We present the step-by-step processes involved in both the client and server components, which are vital for ensuring secure and efficient communication and malware identification. From the client's perspective, we discuss tasks such as configuration parsing, file search, encryption, and database logging. On the server side, we present enclave initialization, data buffer management, pattern matching, and database logging. By thoroughly examining the execution life cycle, we aim to provide a comprehensive understanding of the processes that enable secure data transmission, effective malware analysis, and insightful logging for subsequent analysis and system optimization.

The execution life cycle of the server side entails a series of fundamental steps. First, the server initializes the Intel SGX enclave, establishing a secure and trusted execution environment for confidential data processing. The enclave ensures the integrity and confidentiality of the server's operations, safeguarding against unauthorized access or tampering.

Upon enclave initialization, the server allocates and initializes a data buffer, providing a protected and isolated space for temporary data storage during the analysis process. This buffer is meticulously designed to efficiently handle incoming data and facilitate subsequent operations, ensuring optimal performance and secure data handling.

Furthermore, the server loads the Deterministic Finite Automaton (DFA), a specialized data structure utilized for efficient pattern matching and malware detection. Constructed based on known malicious patterns, the DFA serves as a reference model for identifying potential threats within the analyzed data. Its implementation allows for fast and reliable pattern matching, enhancing the server's capability to detect and mitigate potential security risks. Subsequently, the server establishes a TCP socket and awaits for incoming client connections and malware detection requests.

Once the server is initialized, the client(s) can begin to offload data for malware analysis. The execution life cycle of the program on the client side adheres to a of operations. Initially, the client retrieves the configuration file, parsing its contents to determine pertinent parameters such as the hash mode, encryption bit mode, time interval, iterative file search behavior, database table reset policy, and file quarantine preferences. Armed with this information, the client proceeds to create a TCP socket, and establishing a communication channel with the server.

Upon successful socket creation, the client engages in a secure key exchange protocol with the server, guaranteeing the confidentiality and integrity of subsequent communication. Then, the client initializes a file structure, a data entity designed to encapsulate crucial file attributes.

Then, the client initiates the file iteration phase by conducting an exhaustive search of designated directories and subdirectories, collecting file metadata for analysis. For each encountered file, the client records its path, hash value, and unique identifier within the file structure. Additionally, the client evaluates the file's length and content, employing cryptographic hash functions to calculate the hash value.

Moreover, the client applies data encryption to protect the file contents, ensuring its confidentiality during transit. The encrypted file data is securely transmitted to the server via the established TCP connection. Simultaneously, the client maintains an auditable log in the database, documenting essential information, including file details and timestamps. Following the transmission, the client awaits the server's response, facilitating bidirectional communication and synchronization. The client logs client processing times and database query times into the database, furnishing valuable insights into the execution performance and interaction of the client with the database.

The server continuously receives the incoming data from the socket until the

data buffer reaches its capacity or all the required data has been received. This iterative reading process ensures data integrity and minimizes the risk of data loss or corruption, guaranteeing reliable data transfer between the client and the server.

Once the complete data has been acquired, the server transfers the data within the enclave where they are decrypted with the appropriate shared key. Then, it invokes the matcher function, utilizing the loaded DFA to perform pattern matching and identify potential malware within the received data. Leveraging our custom implementation of the Aho-Corasick algorithm, the matcher function systematically compares the data against the DFA, enabling the accurate detection of potential threats.

Upon completion of the analysis, the server logs relevant information into the database. This includes the total number of matches found and the corresponding files in which they were detected, providing valuable insights for subsequent analysis and investigation. Additionally, the server records crucial statistics, such as processing times, to assess performance and optimize the efficiency of its operations. The server also captures database querying times, facilitating performance analysis and enabling optimization of database interactions for enhanced overall system efficiency.

Once the malware detection is completed, the server notifies the client that the process is completed. Then, the client can react to the identified threats and utilize is quarantine capabilities. In subsequent iterations, the client verifies if the calculated hash value of the marked files differs from the one computed in the previous iteration and requests malware analysis only for the modified files, minimizing the need for data transmission and malware scanning.

In essence, the client's execution life cycle encompasses a series of operations, spanning configuration parsing, secure communication establishment, file retrieval, metadata collection, hash calculation, encryption, transmission, and logging. The server's execution life cycle encompasses critical phases, including enclave initialization, data buffer allocation, DFA loading, socket creation, iterative data reading, pattern matching using the DFA, database logging of match results, and performance logging. These steps ensure the secure analysis of data, efficient malware detection, and provide valuable insights for subsequent analysis, optimization, and security enhancement. By utilizing SGX enclaves on the remote server, we ensure that the malware analysis code, virus signatures and transmitted user data always remain protected and unhampered. Also, we ensure user data privacy since even honest-but-curious infrastructure providers, hosting our server, can never gain access to the transmitted data or modify the signature DFA to perform side channel attacks that could compromise data privacy.

Chapter 5

Evaluation

In this chapter, we present our system's evaluation using micro- and macro- benchmarks and a combination of real and synthetic workloads. We present the performance metrics collected by measuring each component's various operations, as well as end-to-end performance metrics.

5.1 Experimental Setup

Our malware detection server is hosted on a desktop computer with an eight-core Intel i7-8700K CPU, operating at a frequency of 3.7GHz, and is able to utilize Intel SGX enclaves. The server is also equipped with 32GB of DDR4 RAM clocked at 2400MHz. The host is running Arch Linux with Linux kernel version 6.1.29-1-lts and the official Intel SGX SDK, PSW and driver.

The client is hosted on a Raspberry Pi 4, a single-board computer widely used for various applications. The Raspberry Pi 4 features a Broadcom BCM2711 quad-core Cortex-A72 (ARMv8) 64-bit SoC operating at a frequency of 1.5GHz. The client system is equipped with either 2GB, 4GB, or 8GB of LPDDR4-3200 SDRAM. The Raspberry Pi 4 has built-in Gigabit Ethernet for network connectivity. The client and the server reside in the same network for testing purposes.

5.2 Workloads

To thoroughly evaluate our system, we use a combination or real and synthetic workloads. This combination of workloads allows us to obtain the real performance metrics of our system as well as stress the malware detection system on purpose in order to understand its behaviour under the worst case conditions.

Synthetic Workload The synthetic signature set consists of custom patterns, with each pattern being 255 bytes in size. In total, we generate three synthetic signature sets, containing 2000, 4000 and 6000 patterns respectively. In this way, we aim to understand how the number of malware patterns affect the number of

generated DFA states and its overall size, and how, consecutively, this affects the malware detection performance.

Along with this synthetic signature set, we generate a set of six synthetic data workloads. These data sets contain thousands of files (of various data types) ranging in size from 100 KB to 1 MB, aiming to emulate real-world files found in common IoT devices. Each of these data workloads are carefully infected at 0%, 5%, 10%, 20%, 50%, and 100% respectively, using the custom malware patterns. In total, we generate 18 data workloads spanning three sets, each covering various contamination cases for each of the three synthetic signature sets.

The first workload contains only benign files; a case which is the most commonly expected. The next three workloads aim to emulate low, medium, and high infection rates that might be encountered in real use cases. Finally, the workloads infected by 50% and 100% represent cases that we do not expect to encounter but are used to stress the limits of our malware detection engine.

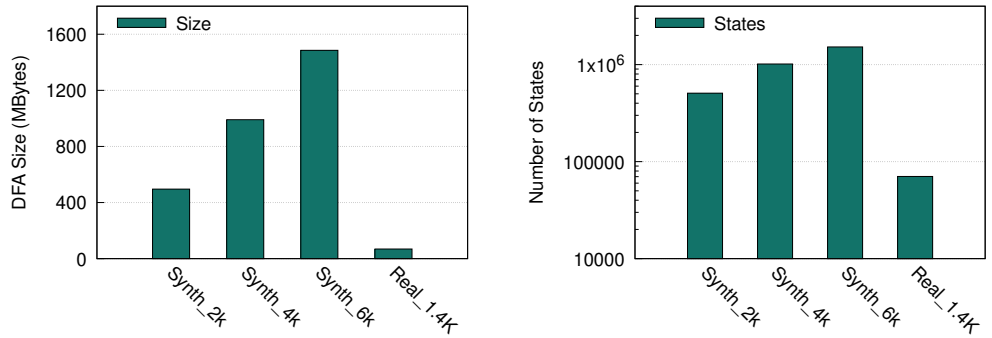
Real Workload In contrast to the synthetic signature workload, the real-world signature workload contains 1400 malware patterns that we acquire and extract from various sources, such as malware signature databases and threat intelligence networks. The patterns in this workload do not have a fixed size and span from several bytes to a few hundreds of bytes. The purpose of this workload is to evaluate our system using real-world data.

To generate the data workload for the real-world ruleset, we gather 1500 benign files from the client's file system, including various files and executables, ranging from 100 KB to 1 MB, for consistency with the synthetic workload. Similar to the synthetic workload, we use these data to generate six workloads, infected by 0%, 5%, 10%, 20%, 50%, and 100% respectively. During the datasets' infection, we select the malware patterns randomly but avoid to infect multiple files with the same pattern, thus triggering all 1400 malware patterns when using the 100% infected workload.

5.3 DFA Properties

We start our evaluation by presenting the properties of the DFAs generated using the three synthetic signature sets (containing 2000, 4000 and 6000 patterns respectively) and the real-world signature set containing 1400 malware patterns. We compile all the signature sets and measure the number of generated states and the overall serialized DFA size. The results of this evaluation are presented in Figure 5.1, where sub-figure 5.1(a) depicts the DFA's size and sub-figure 5.1(b) the number of generated DFA states.

We notice that the DFA size increase between the automaton containing 2000 signatures and 4000 signatures is 100%. Also, the DFA compiled with 6000 malware patterns has a 50% increased size compared to the automaton containing 4000. This indicates that the size of the serialized DFA is proportional to the number



(a) DFA Size. (b) DFA States

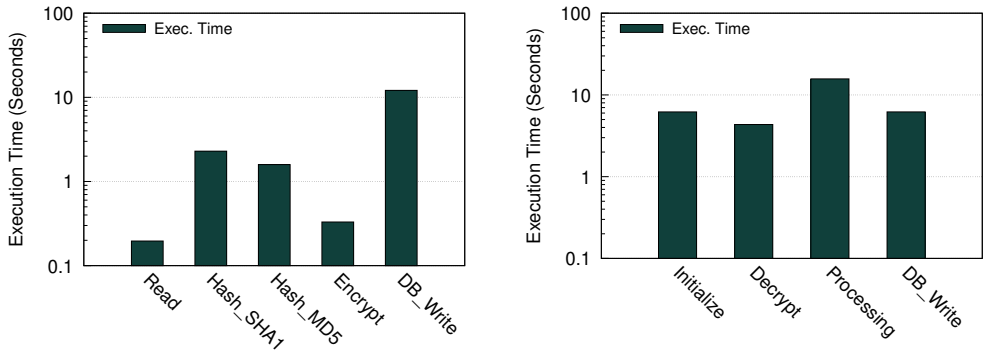
Figure 5.1: DFA size and number of states for the three synthetic signature sets and real-world signature set.

of malware patterns, when they have the same or similar size. Also, we notice that when compiling more than 4000 patterns, sized at 255 bytes, the serialized automaton exceeds 1GB in size. Observing sub-figure 5.1(b), we notice that the same behaviour is reported for the number of generated states.

On the other hand, the DFA compiled with 2000 synthetic patterns has only 40% more patterns than the real-world DFA but requires 7.2 times more memory and has 7.2 times less states. This indicates that the size and randomness of the malware patterns are more important than their absolute number in affecting the DFA's properties. We also notice that the states of the DFA compiled with the real-world signature set are more dense, meaning that each state contains more transitions to other states; a behaviour that is exhibited due to the partial similarity of certain patterns (e.g., common prefixes or suffixes). These results indicate that several thousands of real malware signatures could be compiled to a singe DFA without requiring excessive memory capacity for their storage. Also, this can lead to several optimizations where patterns with similarities can be grouped together to generate multiple small DFAs with better caching properties.

5.4 Micro-benchmarks

In this section, we evaluate the performance of the various operations performed both by the client and the server. At the client's side, we measure the time required to fetch the files that need to be sent remotely for malware analysis and the time required to evaluate their hash values, using both SHA1 and MD5. Also we measure the time required to encrypt them before transmission, using AES-GCM-128 as well as the time required for the client to perform the database logging. The data set used for this evaluation contains ∼1GB of data, spanning 2000 files and the server is configured with the DFA produced by compiling 2000 random synthetic malware signatures. The results of this evaluation are presented in Figure 5.2(a).



(a) Client (b) Server

Figure 5.2: DFA size and number of states for the three synthetic signature sets and real-world signature set.

We notice that the most time consuming operations is the database logging functionality, requiring ∼11 seconds to complete. During this process, the client saves all the required metadata for each transmitted file, such as its name, absolute path, hash values, status, etc. Also, we notice that, as expected, the encryption operation is quite fast, requiring less time to complete compared to the hash value generation. Moreover, MD5 is ∼30% faster than SHA1, with SHA1 providing higher collision resistance. However, MD5 is a perfect alternative for low power IoT devices offering adequate collision resistance for our use case.

We also measure the various operations performed by the malware detection server and report their execution times in Figure 5.2(b). As expected, the malware scanning process consumes the most time, as it is the most complex operation. Also, we notice that data decryption within the SGX enclave is more time consuming than the encryption performed by the client. Moreover, the server performs less write operations to the remote database, as its main logging concerns only the malware detection process, producing less metadata. Finally, the server's initialization time is reported at ∼6 seconds. During this operation, the server sets up all the required data structures, such as the data buffers, initializes its network sockets and SGX enclave, and loads the DFA into the enclave. While this process seams time consuming, we note that it is only performed once, during its bootstrap phase, and is not performed between every consecutive remote malware detection request.

5.5 Malware Detection Performance

In this section, we conclude our evaluation by measuring the sustainable throughput achieved by our enclave-protected malware detection engine. To identify its performance characteristics we evaluate the engine using both the synthetic and the real-world workload.

5.5.1 Synthetic Workload

We begin the evaluation starting with the synthetic workload. For this analysis, we set up the malware detection server, each time using one of the three synthetic signature DFAs, containing 2000, 4000 and 6000 patterns respectively. Then, we utilize the client to offload each synthetic data workload for malware analysis 10 times and report the average sustainable throughput achieved by the malware detection engine. Between each consecutive offload request, we do not place malicious files in the quarantine after their identification so that the data stream maintains its worst case properties.

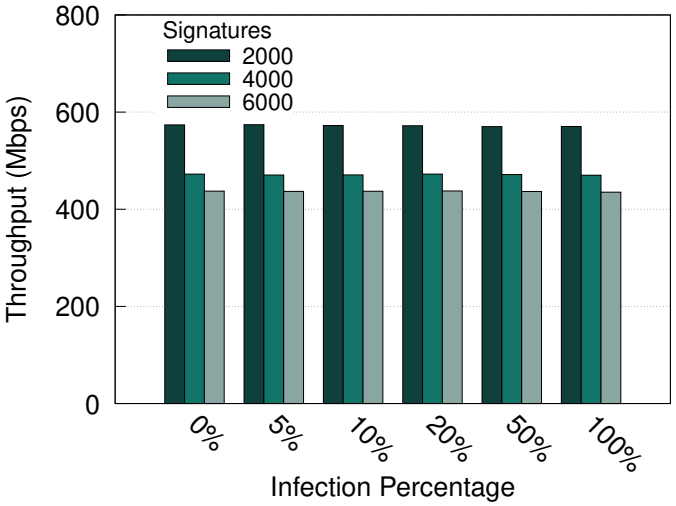


Figure 5.3: Malware detection throughput evaluation using workloads infected by 0%, 5%, 10%, 20%, 50% and 100% and synthetic malware signature sets containing 2000, 4000 and 6000 patterns.

The results of this analysis are presented in Figure 5.3. The X-axis indicates the infection percentage of each data workload while the Y-axis indicates the achievable malware detection throughput in Mbits/second. We notice that the best performance is achieved when utilizing 2000 malware signatures, where the engine is able to perform the analysis at ∼570 Mbps. Increasing the number of patterns to 4000 yields a ∼17.5% decreased throughput. Also, the DFA containing 6000 patterns yields a ∼23% decreased processing throughput compared to the automaton containing 2000 patterns and exhibits a ∼7% performance decrease compared to the DFA compiled with 4000 patterns. These results indicate that the malware scanning performance decreases when the DFA's size increases, but it is not directly proportional to its size.

5.5.2 Real-World Workload

We conclude our evaluation by performing a similar malware detection throughput evaluation, this time using the real-world signature set and data workloads, following the same configuration and methodology as described in the previous section. The results of this analysis are presented in Figure 5.4. The first thing that we notice is that the infection percentage has a low effect on the malware detection engine's performance. The best throughput is achieved when the entire data set is benign, with a ∼0.5% performance decrease when the data are infected at 5% (i.e. every 100 files, 5 are infected). The worst performance is recorded for the 50% and 100% infection rates (i.e., half and all files are infected), yielding ∼1% lower throughput than the benign data set.

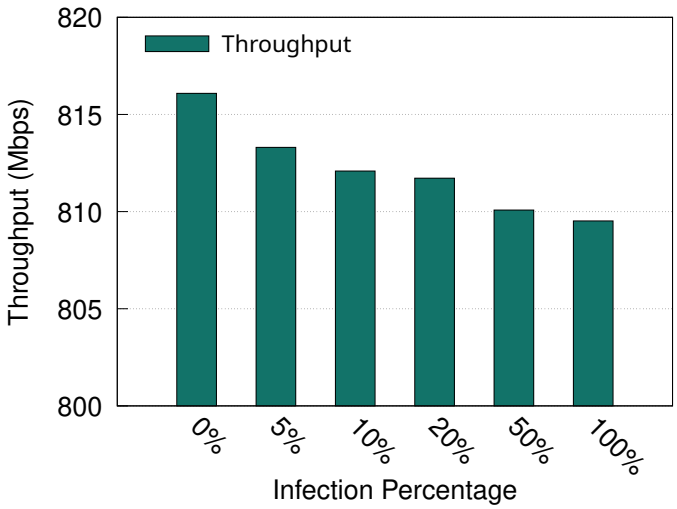


Figure 5.4: Malware detection throughput evaluation using workloads infected by 0%, 5%, 10%, 20%, 50% and 100% and a signature set containing real-world malware patterns, collected by various online sources.

After examining the DFA access patterns during the malware detection process, we notice that the malware engine performs several traversals on the DFA but without managing to reach a final state. This happens due to the high byte randomness exhibited both by the contents of the executable files and the malware signatures, constantly triggering the pattern matching process but without terminating it since the patterns do not always exist in the files. For example, several five- to ten-byte sequences in the input files trigger the prefixes of various patterns. This behaviour eliminates possible caching effects, affecting the overall performance of all workloads. The best caching effects are reported for text-based files, where the possible input bytes are limited to the printable ASCII set, thus triggering less patterns and with smallwe sequences (i.e., for up to five bytes). Finally, we notice that the malware engine is able to achieve up to ∼815 Mbps of malware detec-

tion throughput, being able to process data received by a 1Gbps network channel almost in real time. This results indicate that with a certain level of parallelism and optimizations, the system will be able to sustain real-time processing when communicating over 1Gbps network streams.

Chapter 6

Related Work

The security research community is primarily focused on developing malware detection techniques to safeguard users and organizations from ever-evolving cyber threats. Among the notable contributions in this field, ClamAV [12] has emerged as a popular open-source anti-malware solution that effectively accelerates scanning and matching processes. CloudAV [13] introduced the concept of cloud-based malware scanning, later extended to the mobile environment in [14]. However, while CloudAV achieves high detection rates, it compromises user privacy by exposing sensitive information.

SplitScreen proposes a distributed anti-malware system that utilizes bloom filters to expedite malware scanning [15]. RScam offers another cloud-based anti-malware system that ensures efficient security services and data privacy protection for resource-constrained devices [16]. However, RScam assumes a trusted server environment. In our work, we propose a cloud-based malware detection engine that leverages hardware-assisted enclaves to safeguard user data and preserve privacy.

In the context of cloud-based services, there is a growing interest in outsourcing network processing applications to improve cost-effectiveness, performance, and scalability. APLOMB enables the outsourcing of enterprise middlebox processing in the cloud [17], while BlindBox and Embark propose processing encrypted traffic for confidentiality [18, 19]. The lack of transparency regarding user data manipulation by cloud service providers has been addressed by works like CloudFence [20].

TEEs, such as Intel SGX, have gained attention for ensuring data and code protection. Several works explore the utilization of TEEs for outsourced applications in the cloud, including privacy-preserving data analytics [21, 22, 23] and secure middlebox functionality [24, 25, 26, 27]. Unlike these works, our system aims to provide a secure cloud-based anti-malware solution for all device types while prioritizing user privacy. Furthermore, our approach avoids executing the antivirus solution on top of SGX using unmodified applications, as it would increase the trusted computing base (TCB), potentially enlarging the attack surface and reducing end-to-end performance [28, 29, 30, 31, 32].

In the context of IoT devices State of the Art [33] aims to develop a hybrid

in-cloud malware analysis and detection system for intelligent IoT devices, incorporating a lightweight anti-malware engine, a lightweight agent, and cloud-based anti-malware engines to enhance response time, client protection, and bandwidth efficiency compared to existing systems while EPMDroid [34] is a research endeavor that proposes EPMDroid, a privacy-preserving malware detection scheme utilizing hardware-based Trusted Execution Environment technologies like Intel SGX, to effectively detect mobile malware in the rapidly expanding mobile communication networks and IoT, overcoming memory limitations through data fusion techniques.

Several recent advancements have emerged to enhance the security of SGX, focusing on mitigating memory bugs [35, 36]. SGXBOUNDS [37] introduces efficient bounds-checking mechanisms that minimize memory overhead and accommodate the limited size of the EPC. SGX-Shield [38] implements Address Space Layout Randomization (ASLR) within enclaves, employing a scheme to maximize entropy and provides the capability to conceal and enforce ASLR decisions. Eleos [39] presents a novel approach to reduce enclave exits by asynchronously servicing system calls outside of enclaves and enabling user-space memory paging. Additionally, SGX-Elide [40] aims to protect the confidentiality of SGX code itself by enabling dynamic updates of the enclave code. Lastly, T-SGX [10] combines SGX with Transactional Synchronization Extensions to mitigate controlled-channel attacks. These advancements operate independently from our security stack and can be seamlessly integrated into our proposed system.

Chapter 7

Conclusion and Future Work

In conclusion, this work presents the developement of a cloud-based malware detection system, aiming to provide malware scanning as a service to the IoT ecosystem. Our work highlights the potential of user-level enclave-based Trusted Execution Environments (TEEs) in developing a secure and privacy preserving malware detection solution that safeguards the protection tool's execution as well as the privacy of the offloaded data, even on hostile infrastructure.

By securely offloading the computationally intensive malware analysis task to remote servers, hosted within SGX enclaves, the proposed system addresses performance limitations enforced by low-power IoT devices and provides a scalable and centralized protection solution. The system optimizes IoT resource utilization and offers a new approach to signature-based malware detection in IoT environments.

The research outcomes aim to contribute to the advancement of IoT security by providing insights into preserving privacy and security in interconnected devices. Additionally, the proposed system enables secure communication, attestation, enhanced data privacy and auditable logging capabilities. Also, in this work we explore customizations that can be performed to popular pattern-matching algorithms used in the field of malware detection, such as Aho-Corasick, that enable them to be efficiently and securely executed in memory-constrained TEEs.

7.1 Summary of Contributions

In summary, the contributions of this work are the following:

- We propose a practical cloud-based malware detection solution that aims to provide malware detection service to a plethora of IoT devices without exhausting their limited resources.

- Our system prioritizes strong privacy-preserving guarantees for clients with emphasis on the secure remote analysis of their sensitive data.

- We present the methodology that can be used to leverage user-level enclaves in designing a signature detection engine able to preserve its security properties even when residing in untrusted or hostile infrastructure.

7.2 Future Work

As future work, there are several areas that can be explored to further enhance our proposed cloud-based malware detection solution leveraging Intel SGX enclaves. First, we aim to optimize our server for handling multiple clients concurrently. By implementing efficient thread handling mechanisms, we can improve the scalability and performance of our system allowing it to handle a larger number of IoT devices simultaneously. This can be particularly beneficial in scenarios where there is a significant number of IoT devices requiring malware analysis at the same time. Also, we plan to implement load balancing mechanisms responsible for managing multiple server instances, aiming to mitigate possible load spikes and latency issues when handling large workloads.

Second, we can integrate advanced threat intelligence capabilities into our solution. This can involve incorporating YARA rules, a powerful format for identifying and categorizing malware based on patterns and characteristics. We can automate the retrieval of YARA rules from threat intelligence networks using STIX (Structured Threat Information Expression), which will provide up-to-date and relevant rules for malware detection in a completely automated manner. This integration can greatly enhance the accuracy and effectiveness of our malware detection system, enabling it to detect and mitigate new and emerging threats and immediately respond to malware campaigns.

Additionally, exploring the possibilities offered by Intel SGX2, the next generation of Intel SGX technology, holds promise for future enhancements. SGX2 introduces new features and improvements, such as larger enclave sizes, enhanced attestation mechanisms, and support for dynamic memory management within enclaves. By leveraging SGX2, we can further optimize the performance and security of our solution, enabling even more efficient and robust malware detection capabilities and the ability to store and process several thousands of malware patterns simultaneously while serving multiple clients concurrently.

Bibliography

[1] "Knuth–morris–pratt algorithm," https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm.

[2] "Boyer–moore string-search algorithm," https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm.

[3] "Rabin–karp algorithm," https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm.

[4] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[5] "Intel sgx," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/get-started.html.

[6] "Arm trustzone," https://www.arm.com/technologies/trustzone-for-cortex-a.

[7] "Amd sev-snp," https://www.amd.com/system/files/techdocs/sev-snp-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[8] "Amd psp," https://en.wikipedia.org/wiki/AMD_Platform_Security_Processor.

[9] "Apple secure enclave," https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web.

[10] "Enclavedb," https://www.microsoft.com/en-us/research/uploads/prod/2018/02/enclavedb.pdf.

[11] "Diffie-hellman," https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.

[12] "Clamav-cisco talos intelligence group," https://www.talosintelligence.com/clamav.

[13] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud." in *USENIX Security Symposium*, 2008, pp. 91–106.

[14] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, "Virtualized in-cloud security services for mobile devices," in *Proceedings of the first workshop on virtualization in mobile computing*, 2008, pp. 31–35.

[15] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, "Splitscreen: Enabling efficient, distributed malware detection," *Journal of Communications and Networks*, vol. 13, no. 2, pp. 187–200, 2011.

[16] H. Sun, X. Wang, J. Su, and P. Chen, "Rscam: Cloud-based anti-malware via reversible sketch," in *Security and Privacy in Communication Networks: 11th EAI International Conference, SecureComm 2015, Dallas, TX, USA, October 26-29, 2015, Proceedings 11.* Springer, 2015, pp. 157–174.

[17] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[18] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *Proceedings of the 2015 ACM conference on special interest group on data communication*, 2015, pp. 213–226.

[19] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 255–273.

[20] V. Pappas, V. P. Kemerlis, A. Zavou, M. Polychronakis, and A. D. Keromytis, "Cloudfence: Data flow tracking as a cloud service," in *Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings 16.* Springer, 2013, pp. 411–431.

[21] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE symposium on security and privacy.* IEEE, 2015, pp. 38–54.

[22] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 283–298.

[23] S. Chandra, V. Karande, Z. Lin, L. Khan, M. Kantarcioglu, and B. Thuraisingham, "Securing data analytics on sgx with randomization," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22.* Springer, 2017, pp. 352–369.

[24] C. Priebe, K. Vaswani, and M. Costa, "Enclavedb: A secure database using sgx," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 264–278.

[25] D. Goltzsche, S. Rüsch, M. Nieke, S. Vaucher, N. Weichbrodt, V. Schiavoni, P.-L. Aublin, P. Cosa, C. Fetzer, P. Felber *et al.*, "Endbox: Scalable middlebox functions using client-side trusted execution," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 386–397.

[26] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Shieldbox: Secure middleboxes using shielded execution," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–14.

[27] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "{SafeBricks}: Shielding network functions in the cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 201–216.

[28] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "{SCONE}: Secure linux containers with intel {SGX}," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 689–703.

[29] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, pp. 1–26, 2015.

[30] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves." in *NDSS*, 2017.

[31] H. Tian, Y. Zhang, C. Xing, and S. Yan, "Sgxkernel: A library operating system optimized for intel sgx," in *Proceedings of the Computing Frontiers Conference*, 2017, pp. 35–44.

[32] C.-C. Tsai, D. E. Porter, and M. Vij, "{Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[33] S. Alam, I. Sogukpinar, I. Traore, and Y. Coady, "In-cloud malware analysis and detection: State of the art," in *Proceedings of the 7th International Conference on Security of Information and Networks*, 2014, pp. 473–478.

[34] W. Wei, J. Wang, Z. Yan, and W. Ding, "Epmdroid: Efficient and privacy-preserving malware detection based on sgx through data fusion," *Information Fusion*, vol. 82, pp. 43–57, 2022.

[35] "Fabrice bellard. quickjs benchmark." https://bellard.org/quickjs/bench.html.

[36] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 193–206.

[37] D. Kuvaiskii, O. Oleksenko, S. Arnautov, B. Trach, P. Bhatotia, P. Felber, and C. Fetzer, "Sgxbounds: Memory safety for shielded execution," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 205–221.

[38] J. Seo, B. Lee, S. M. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim, "Sgx-shield: Enabling address space layout randomization for sgx programs." in *NDSS*, 2017.

[39] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, "Eleos: Exitless os services for sgx enclaves," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 238–253.

[40] E. Bauman, H. Wang, M. Zhang, and Z. Lin, "Sgxelide: enabling enclave code secrecy via self-modification," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 75–86.