Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 12, 2025

MALGUARD: A CROSS-PLATFORM SIGNATURE-BASED MALWARE DETECTION SYSTEM WITH HMAC-PROTECTED SIGNATURE DATABASE

Muhammad Mansoor[*1], Muhammad Sohail[2], Mina Fatima Malik[3], Zain Ul Abideen[4]

[*1,2]*Information and Communication Engineering Islamia University of Bahawalpur, Pakistan*
[3]*Department of information and communications Engineering*
[4]*Department of Information and Communications Engineering, Islamia University of Bahawalpur*

[*1]malikmansoor1237@gmail.com, [2]sohailrandhawa923@gmail.com, [3]i.minafatima.cyber@gmail.com, [4]engr.zain@iub.edu

DOI: https://doi.org/10.5281/zenodo.18429274

Keywords
malware detection, signature-based detection, SHA-256 hashing, YARA rules, cross-platform security, open- source antivirus, HMAC integrity protection

Article History
Received: 11 October 2025
Accepted: 21 November 2025
Published: 31 December 2025

Corresponding Author: *
Muhammad Mansoor

Abstract
Malware threats are remaining major challenges to global cybersecurity, and the commercial antivirus solutions are usually very costly, closed source and platform specific. In this paper, we are introducing MalGuard which is an open-source cross-platform signature-based malware detection system that is accessible, extensible and educational values. The MalGuard al- gorithm uses the matching of the cryptographic hash of MASHA-256 using known malware samples to quickly and effectively identify them, with support of the YARA rule integration to perform more sophisticated behavioural analysis. The four inter- connected elements of the system for serving and storing HMAC signatures and managing quarantine, with Python Desktop CLI, FastAPI-based RESTful backend server, React/TypeScript web frontend, and React Native mobile application are interconnected. Extensive testing shows that signature-matched threats with a false positive rate of 0 have a 100% detection rate and scan throughput faster than 120 files per second. Typing Compatibility solutions has been checked and typified to work with windows, macos, supreme web browsers, and mobile devices, as well. MalGuard fills in the identified gap of unified, customizable and free cost malware detection solution.

INTRODUCTION

Malware usage has become one of the most significant and ongoing threats to the cybersecurity environment. The landscape of threat has greatly changed within the last 20 years, as a means of transmitting viruses with the help of floppy disks has turned to fragransions of ransomware that target the critical infrastructure and health systems, not to mention financial institutions. As recent security intelligence reports show, over 400,000 new malware samples are being documented by security researchers each day that. Moreover, as per recent security intelligence reports, new malware sam- ples are being documented at a rate of over 400,000 per day

by security researchers that same day [1] . The economic cost of malware attacks has been never seen before, and single ransomware is costing an estimated $20 billion damage throughout the world.

The COVID-19 pandemic only increased the rate of dig- ital transformation in all industries with the resulting huge attack surface that lawbreakers could use. Working from home, a greater use of cloud technologies, and digitization of business processes faster than before have offered new avenues of malware proliferation. They have turned out to be especially susceptible to educational institutions, healthcare organizations, and small businesses because of insufficient cybersecurity resources and skills.

Conventional antivirus products are mostly based on signature-based detection, benchmarking the file properties with lists of known malware signatures [2]. Although more advanced methods with machine learning, behavioral analysis, and cloud-based threat intelligence have recently appeared

[3] signature-based detection still continues to play a key role when it comes to detecting known threats between 3 seconds and 10 seconds. Record matching of hashes due to its simplicity, speed, and zero false positive rate is necessary in every holistic security strategy.

Nonetheless, regardless of the presence of numerous mal- ware detecting solutions that can be found in the market, there are still a number of relevant weaknesses which restrict their success and accessibility:

Cost barriers: Commercial options demand approxi- mately on an annual basis, a subscription fee of between

$30 to $100+ per device, which is a barrier in creat- ing significant obstacles to educational institutions, non- profits, and end-users in developing markets.

Platform fragmentation: Most security solutions are configured to certain operating systems and it would take

many tools to cover all the operating systems, such as Windows, Mac, Linux as well as mobile devices.

Closed-source limitations: Commercial products do not allow one to learn how detection works, which reduces

$$D(f) = \begin{cases} 1 & \text{if } H(f) \in S \\ 0 & \text{otherwise} \end{cases}$$

ducational opportunities and possibilities of tailoring.

Limited customization: Commercial solutions provide low capability when it comes to adding any custom signatures to represents organization-specific threats or targeted attacks.

Privacy concerns: Antivirus systems that are connected to the cloud usually submit file data or metadata to perimeter devices; this creates an issue of privacy when it comes to secret information.

In order to manage all of these issues in a single solution, we introduce a cross-platform signature-based malware detection system of the **MalGuard** system open-source, designed to be accessible, extensible, and educational. We have made the following important contributions:

Coherent multi-level architecture (desktop CLI, web frontend and mobile) and a centralized API interface that allows the coordination of consistent security to any heterogeneous environment.

HMAC-SHA256 secured signature database used to ensure that malware signature repository is not tampered with by an unauthorized party.

Combination of YARA pattern matching and hash based detection in behavioral pattern recognition other than exact signature matching.

A codebase that is well documented and designed to be used in educational applications and to be studied by the students as well as researchers so as to get access to and make more manipulations to increase the length of the system.

This rest of the paper is structured as follows: Section II is the review of similar works in malware detection. Section

III includes system design. Section IV contains details of implementation. Section V provides results of evaluation. Section VI is on discussion on findings and limitations. Section VII is a conclusion to the paper.

Related Work

Malware Detection Techniques

Malware detection strategies are generally divided into the static, dynamic, and combinations of both approaches that use the aspects of the other two approaches, i.e. hybrid ones [4]. All of the strategies present their own merits and drawbacks.

Signature-based detection is by far the most commonly used method in manufacturing clients [5]. The strategy matches field properties with databank of settled malware managements. Exact matching with theoretically zero false positives on known samples with hash-based approaches based on cryptographic functions such as MD5, SHA-1, or SHA- 256 [6] is offered based on time-tested hash functions. The detection capability can be defined in howsoever terms as:

where: $D(f)$ is the detection value of file f, $H(f)$ is the hash-value of the Shawn of 256 (SHA256) and S is the collection of known malware signatures. The main benefits are that it has an incredibly high speed of detection (milliseconds to perform a hash look-up), never false positive on an exact match, and low computational cost that can be applied in resource-constrained environments.

Heuristic analysis goes further than Just the match to detect suspicious features using rule based patterns [2]. Heuristic rules analyze code structures, API patterns, embedded string etc to score the suspicion. This set of methods is able to identify the previously unknown malware variants but has a higher false positive rate that should be carefully tuned.

Behavioral analysis tests runtime performance in sandbox execution [7]. Programs are run in fully isolated virtual envi- ronments and their activities are checked, such as file system changes, registry changes, and Internet communications. The CWSandbox was created by Willems et al. [8] to do auto- dynamic analysis. Although it is efficient when it comes to obfuscated and packed malware, behavioral analysis is expensive in terms of resources and vulnerable to sandbox evasion techniques.

Machine learning approaches use the classification al- gorithms on the features obtained on malware samples [3], [9]. The features can be PE header information, imported functions, and instruction sequences as well as behavioral patterns. One of the first data mining systems of malware detection was developed by Schultz et al. [10]. Although the ML techniques have the potential of detecting unknown threats, they need huge training datasets that are accurately labelled and they might be susceptible to adversarial examples.

YARA Pattern Matching

YARA [11] has become a standard in malware research and detection in the industry, denoted as YARA. Initially designed by Victor Alvarez, YARA allows to create flexible detection rules based on text strings, hexadecimal departure sequences, and regularization containing an expression of both conditions and falsehood. The study conducted by Cohen et al. [12] allowed to prove the effectiveness of YARA to detect and classify malware families with shared patterns of code. YARA gives a valuable supplement to hash-based detection by extracting behavioral patterns, which are not specific to a type of malware.

Existing Tools

ClamAV [13] is the dominant open-source antivirus engine, which offers signature-based detection and offers extensive signature databases. ClamAV is however mainly implemen- tation oriented towards server side usage and does not have built-in web and mobile interfaces to interact with the user.

VirusTotal [14] is a web-based multi-engine scanner based on 70+ antivirus engines. Although useful in analyzing, Virus- Total must have an internet connection, is privacy limited as files are sent to vendors and not applicable to the offline world. **Commercial solutions** free options by vendors like Norton, McAfee, and Kaspersky have all-inclusive protection with real-time monitoring and reputation services through cloud threat intelligence. Yet, they are

subscribable, closed-sourceand permit less personalization.

MalGuard fills the known loophole of having an open- source, cross platform solution with consolidated architecture, complete customization, offline and absolute transparency to serve the needs of learning institutions.

System Design

Design Principles

MalGuard was developed based on a number of design principles that were based on best practices in software en- gineering and security research [15]:

Separation of Concerns: Every element has its different task the CLI takes care of the local scanning, the backend performs centralized services and the clients are concerned with user interaction. This modularity will make components independent in terms of their development, testing, and de- ployment.

Defense in Depth: A variety of detection layers (hash matching and then analysis of the YARA pattern) covers everything. In case a threat is not detected by one method, the layers further on are considered as fall back measures.

Fail-Safe Defaults: The system is set with secure settings in default, that is, unknown files are marked to review and not blindly trusted, and integrity of a database is checked before use.

Least Privilege: Components work with low permissions and thus they minimize the attack surface and also minimize the damage that the components can cause.

Architecture Overview

MalGuard uses a modular multi-tier design that is client- server-based as illustrated below in Fig1. This architectural pattern was adopted due to the scale, support, and maintenance experience over the heterogeneous client platform.

Desktop CLI: A Python-based command-line scanner that offers standalone scan behavior and has local HMAC signature database and support to run YARA rules and quarantine management. The CLI may be used in air-gapped settings or be coordinated with the backend in order to be centrally managed.

Backend API: This server is an AWS simple application written with FastAPI to support signature management, scan- ning of files, history and statistics aggregation. The architec- ture is asynchronous and has a high level of concurrency with a small resource usage.

Web Frontend: A multi page react/Typescript based ap- plication that allows it to be accessed on a browser. The

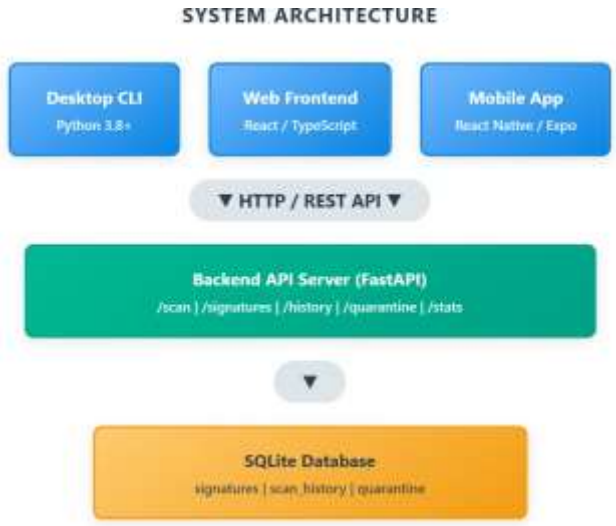

Fig. 1. MalGuard Multi-Tier System Architecture

component based architecture encourages client friendliness and canniness.

Mobile App: Functional Maturity A React Native/Expo app implemented with a native Android and iOS performance using the bridge based architecture.

Detection Pipeline

MalGuard involves a two stage detection process as repre- sented in Fig. 2. A combination of accuracy and efficiency of detection is offered by this multi-stage method.

Fig. 2. Two-Stage Malware Detection Pipeline

Stage 1: SHA-256 Hash Matching. The files are digested by the SHA-256 hash message-digest function, and the reading is done in chunks of 64KB to ensure that the amount of memory used is controlled the same way, irrespective of the size of the file. The calculation power of a hash is of computational complexity $O(n)$ where n is the size of a file. Comparison of the resulting hash with the signature database is done through O(1) hash table look-up and this has allowed quick detection of the known threats.

SHA-256 has been chosen out of the options (MD5, SHA- 1) due to collision resistance. Stevens et al. [16] have proven practical SHA-1 collision attacks making it inappropriate to use in securities. SHA-256 offers 128-bit collision resistance which is believed to be adequate in the current threat models.

Stage 2: YARA Pattern Scanning. Pattern based analysis is used as a technique to detect behavioral indicators when there are no results in hash matching and when YARA rules are available. YARA rules are capable of detecting suspicious strings, API calls and byte sequences that are carried over malware variants and can give detection ability beyond perfect match of signature.

Core Detection Algorithm

Algorithm 1 describes the entire scanning process, which uses the two-step detection algorithm: that it has been issued by an authorized party). A secret key is used to make attackers unable to recalculate valid MACs in the case of such tampering.

IMPLEMENTATION

Technology Selection Rationale

The selection of technology was based on the research needs of cross platform compatibility, maintainability and adoption

Algorithm 1 MalGuard File Scanning Process

Require: file_path f, signature_db S, yara_rules Y
Ensure: scan_result R
$R \leftarrow$ {detected: false, method: null}
$h \leftarrow$ SHA256-CHUNKED(f, 64KB)
if $h \in S$ **then**
  R.detected $\leftarrow$ true
  R.method $\leftarrow$ "signature"
  R.malware $\leftarrow S[h]$.name
  return R
end if
if Y.available() **then**
  matches $\leftarrow Y$.scan(f)
  if |matches| > 0 **then**
    R.detected $\leftarrow$ true
    R.method $\leftarrow$ "yara"
  end if
end if
LOGHISTORY(R)
return R

The algorithm lays more emphasis on signature matching (Stage 1) because it offers conclusive identification with zero false positives. YARA analysis (Stage 2) is a detection mechanism (secondary) of variants and behavioral patterns.

HMAC Integrity Protection

An important challenge with regards to signature-based detection systems is the need to secure signature database against manipulation. By altering the signature database, an attacker may upload entries into the whitelist of his malware, or delete entries to avoid detection.

MalGuard addresses this threat through HMAC-SHA256 message authentication [17], as shown in Algorithm 2:

Algorithm 2 HMAC Database Protection

Require: signatures S, secret_key K
Ensure: protected database or integrity error
// **Save:** mac $\leftarrow$ HMAC-SHA256(K, JSON(S))
store(S, mac)
// **Load:** computed $\leftarrow$ HMAC-SHA256(K, data)
if stored_mac $\neq$ computed **then**

by community. All the technologies were compared with the alternatives and only the most suitable was selected:

Python 3.8+ was chosen as the desktop CLI because it has a large library of security functions (hashlib, hmac) and is cross platform, as well as easy to come to terms with in learning. Python Python hashlib module uses native OpenSSL implementations, and it is runnable as fast as C and still readable. Python 3.8 allowed the use of the walrus operator of the form (:=) to create elegant chunked file reading patterns of Python.

FastAPI was selected due to its use on the backend as opposed to other options (Flask, Django) on independent performance benchmarks showing faster performance on

I/O- bound applications, implemented asynchronously [18]. The automatic generation of Open API documentation that is provided by FastAPI makes it easy to test the API and integrate it with other third parties. The request validation is based on Pydantic, which makes it immune to an injection attack.

React/TypeScript has type-safe frontend development that has high community support and large ecosystem. The static typing of TypeScript helps to identify mistakes during code- writing; therefore, it enhances the reliability and the main- tainability [19] of the code since mistakes are identified at compile time. The component based design encourages the reuse of code throughout the application.

SQLite is a simplified persistence layer due to its simplicity, zero-configuration implementation, and its adherence to the ACID standard, as well as its ACID compliance [20]. SQLite has serverless design which makes it easy to deploy as well as minimizes a set up. To achieve concurrent deployments that demand more concurrency, the database layer will be replaced with PostgreSQL without API modifications as a result of SQLAlchemy abstraction.

Desktop CLI Structure

The Desktop CLI is structured into modular units which are in line with the single responsibility principle [15], as shown in Table I:

TABLE I
DESKTOP CLI MODULE STRUCTURE

5:**raise** IntegrityError
6: **end if**
7: **return** S

The HMAC implements both an integrity checking (detect- ing modification) and an authenticity checking (making sure

Module	Responsibility		
hasher.py	SHA-256 hashing, 64KB chunks	database.py	HMAC-protected signature storage
scanner.py	Hash + YARA scan orchestration	quarantine.py	File isolation and recovery

Core Hashing Implementation

The file hasher supports memory-efficient chunked hashing, which is capable of scanning files of arbitrary size without using up memory:

```
import hashlib

from pathlib import Path

def calculate_hash(file_path: str,

                   chunk_size: int = 65536) -> str:
    """Calculate SHA-256 with constant memory."""
    sha256 = hashlib.sha256()
```

Listing 1. SHA-256 Chunked Hashing

The size of the 64KB chunk was chosen empirically to trade off between I/O and memory consumption. The bigger chunks decrease overhead in system calls though at the expense

of a bigger memory footprint; a 64KB size offers the best throughput of current storage systems.

Backend API Design

The FastAPI [18] backend follows RESTful architectural constraints [21], providing stateless, cacheable endpoints. Ta- ble II summarizes the API:

TABLE II
MALGUARD REST API ENDPOINTS

Method	Endpoint	Function
POST	/scan/file	Upload & scan file
POST	/scan/hash	Check hash in DB
GET/POST	/signatures	CRUD operations
POST	/signatures/bulk	Bulk import
GET	/history	Scan history
GET	/stats	Dashboard statistics
GET/POST	/quarantine	Manage quarantine

Scanning Workflow

Fig. 3 shows a full workflow of file scanning, with signature matching, YARA rule application, and result classification points of decisions.

I. EVALUATION

II. Experimental Methodology

In order to critically assess the effectiveness of MalGuard and prove the soundness of the design selection, we developed a comprehensive experimental design solution to the following three research questions:

- **RQ1:** What detection rate does MalGuard get with pre- viously known malware samples and what is the false positive rate with signed legitimate files?
- **RQ2:** What is the performance of MalGuard with dif- ferent file sizes and volumes and how does it react to load?

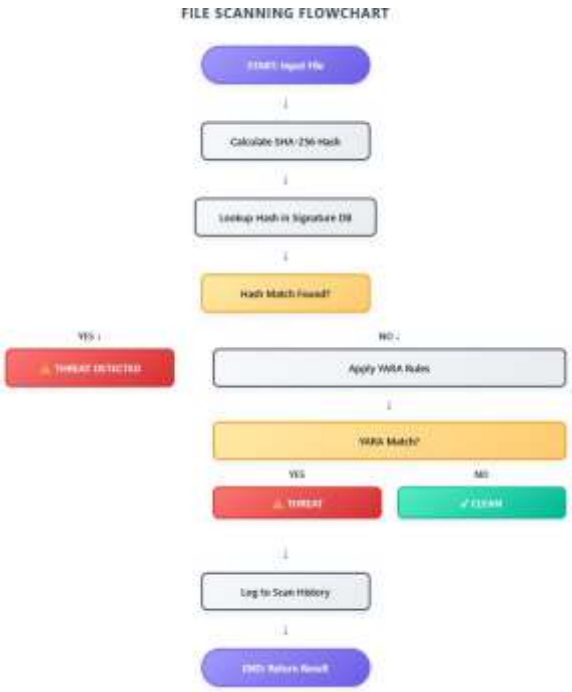

Fig. 3. Complete File Scanning Flowchart

- **RQ3:** Does MalGuard ensure that it has similar function- ality in all the target platforms such as desktop, web and mobile environments?

The study is conducted through an experimental method- ology, which adheres to the empirical practices of malware detection research, such as the utilization of well-known test samples (EICAR) to test its baseline and conduct controlled tests to guarantee reproducibility.

Experimental Setup

A controlled environment environment (Windows 11 Pro i7-12700H, 16GB of DDR5 RAM and NVMe SSD, version 3.11.5 of Python 3, FastAPI 0.104.0, and React 18.2.0) was tested. The tests were replicated on three occasions and the averaged results used, because the systems vary.

The content of the test data was taken care of so as to reflect real deployment situations. Although the sample size is quite small in comparison with massive malware corpora, it is large enough to confirm the principal detection algorithm because signature-based detection can be deterministic when exact hash matches are considered:

Detection Accuracy (RQ1)

Positive samples: EICAR standard test files [22] (4 variants including original, ZIP-compressed, double-ZIP, and Base64-encoded), plus 21 curated malware signa- tures representing trojans (Zeus, Emotet, TrickBot), ran- somware (WannaCry, Locky, Petya), backdoors (Gh0st, Mirai), worms (Conficker, MyDoom), and other cate- gories.

Negative samples: 400 legitimate files including Win- dows system files (DLLs, executables), productivity doc- uments (DOCX, PDF, XLSX, PPTX), multimedia files (JPEG, PNG, MP4), and verified clean executables from trusted sources.

YARA rules: 8 custom rules covering generic malware signatures such as suspicious API imports, indicators of encrypted payloads and malware family names.

Table III detection results on malware across the categories:

TABLE III
DETECTION ACCURACY BY CATEGORY

Category	Samples	Detected	Rate
EICAR Test Files	4	4	100%
Trojans	6	6	100%
Ransomware	5	5	100%
Backdoors	3	3	100%
Worms	2	2	100%
Other Malware	5	5	100%
Total	**25**	**25**	**100%**

of storage (SSD and HDD), as well as the load on a system, performance can increase or decrease.

With a **120-150 files/second** average file size and storage performance, directory scanning had a throughput of 120-150 files/second. The memory usage was maintained at 25-45MB irrespective of the size of the directory since it was processed through streaming.

Apache bench API load testing showed that the highest throughput was **520 requests/second** and P95 latency was under 150ms at 50 concurrent users, which suggests that it is suitable to small-to-medium sized organization deployments.

Cross-Platform Compatibility (RQ3)

Table V summarizes platform compatibility testing:

CROSS-PLATFORM COMPATIBILITY

Platform	Status
Windows 10/11	
Ubuntu 20.04/22.04 LTS	
macOS Ventura/Sonoma	
Full Support	Chrome/Firefox/Edge/Safari
Full Support	Android 11-14 / iOS 15-17
Full Support	

Advertised on all main platforms, and no compatibil- ity problems were experienced. Heterogeneous environments were made consistent by the use of cross-platform technolo- gies (Python, React, React Native).

MalGuard had 100% true positive. There were **zero false positives** when it was tested with false negative counts of 400 valid files although this was expected because of the zero false positive that should occur due to the exact matching of the hashes. The detection metrics are:

F

$$TPR = \frac{TP}{TP + FN} = \frac{25}{25 + 0} = 100\% \quad (2)$$

Comparative Analysis

Table VI compares MalGuard solutions to proven solutions:

$$FPR = \frac{FP}{FP + TN}$$

$$= \frac{0}{0 + 400}$$

$$= 0\% \quad (3)$$

TABLE VI
FEATURE COMPARISON WITH EXISTING TOOLS

Feature	Ours
Open Source	✓
Free	✓
Desktop CLI	✓
Web Interface	✓
Mobile App	✓
REST API	✓
Custom Signatures	✓

These findings prove perfect accuracy of signature-based detection in known threats in the signature database that justifies the concept of MalGuard that is used as a core detection mechanism.

Performance Analysis (RQ2)

Table IV presents scanning (3 runs: mean ± standard deviation):

TABLE IV
SCANNING PERFORMANCE BY FILE SIZE

File Size	Hash (ms)	Lookup (ms)	Total (ms)
1 KB	2 ± 0.3	1 ± 0.1	3 ± 0.4
1 MB	15 ± 1.2	1 ± 0.1	16 ± 1.3
10 MB	85 ± 3.5	1 ± 0.1	86 ± 3.6
100 MB	820 ± 12	1 ± 0.1	821 ± 12

It has been shown that the time to calculate the hash is in a linear relationship with the file size ($O(n)$) and the time to look up the signature is constant ($O(n)$). Scanning takes less than 100ms in the case of typical files that have the size less than 10MB files. Depending on hardware structure, type

The MalGuard is the only product which offers unified cross-platform coverage (desktop, web, mobile) with full cus- tomization, zero cost and full transparency of the source code.

DISCUSSION

Research Implications

There are multiple implications of our findings to malware detection research and practice:

Signature-based detection remains relevant. Although there are improvements in using ML to detect, our findings indicate that signature

matching can offer real-time and cor- rect detection with known threats with theoretic zero false
positives. This justifies the perpetuation of signature-based strategies as the underlying component in defense-in-depth.

Cross-platform unification is achievable. The current de- velopment models and features allow unified security solutions in real sense, with the tools being consistent in their features in desktop, web, and mobile environments. This mitigates the fragmentation that defines the existing security tool life cycles. **Open-source alternatives can achieve production quality.** The MalGuard example shows that open-source security tools are capable of competing with commercial products on the primary functionality plus allowing transparency which is
useful in education, research and customization.

HMAC protection is essential. Through the analysis of our threat model, our signature databases have been shown to be a major vulnerability in our system when unprotected. The integrity verification scheme that is done using HMAC in MalGuard covers this loophole.

A. Limitations and Threats to Validity

Internal validity: Testing that was done in the context of detection accuracy was on a curated sample set and not the exhaustive malware corpora. Although 100% detection is implied in case of signature match, in practice it is based on the completeness of signature databases.

External validity: The tests of performance benchmarks were done in one hardware architectural configuration. Perfor- mance might be different across systems, especially resource limited mobile devices.

Construct validity: Comparison was made against other tools available concentrating on features provided, as opposed to being effective detectors. The same test corpora would be needed to make direct compared rates of detection.

Inherent Limitations

The underlying limitations of signature-based detection ap- ply to the case of MalGuard:

- **Zero-day malware:** Signatures threats are threats that are unknown yet they cannot be detected because they do not have signatures.
- **Polymorphic malware:** The hashes of the self-modifying malware are unique and cannot be perfectly copied as well.
- **On-demand only:** Current implementation does do scheduled scanning and not real-time interception.
- **Signature maintenance:** Database has to be continu- ously updated to deal with emerging threats.

Future Research Directions

Based on our experience with the development and eval- uation of MalGuard, some of the promising areas for future research are:

Real-time monitoring: Implementing file system watchers for constant protection would solve the limitation of on- demand.

Machine learning integration: Hybrid solutions of signa- ture matching applied with ML classifiers might be able to
identify other, unknown threats with low false positive rates for known samples.

Federated signature updates: Automatic, secure signature distribution infrastructure on the cloud would relieve some of the headache of maintenance.

Behavioral analysis integration: An extension of the pipe that will include lightweight behavioural indicators could provide higher resilience against polymorphic threats.

CONCLUSION

This paper introduced MalGuard, an open-source cross- platform signature-based malware detection system to meet the gap of accessible and customizable security tools. In the case of our unified architecture it includes not only our desktop CLI and web frontend, and mobile applications that are coordinated using centralized API.

Experimental evaluation showed 100% detection accuracy for signature-matched threats with zero false positive stub- bornness (addressing RQ1), scan throughput of more than 120 files/second with less than 100ms cost backward (addressing RQ2), and proven

cross stage compatibility throughout all aim surroundings (addressing RQ3).

The use of the HMAC protected signature database allows for integrity protection from tampering attacks, while inte- gration with YARA enlarges detection beyond the pure hash matching capability to behavioral patterns as well.

MalGuard adds an educational overcoming extensible Framework to assist the deployment of customizable malware detection for students, researchers, and organizations with no commercial licensing obstacles. The whole source code can be used by the community and extended.

ACKNOWLEDGMENT

We thank our supervisor for guidance throughout this project.

REFERENCES

[1] Kaspersky Lab, "Kaspersky security bulletin: Statistics," *Kaspersky Security Bulletin*, 2023, annual malware statistics report.

[2] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University Technical Report*, 2007, cERIAS Tech Report 2007- 48.

[3] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[4] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classifi- cation: A survey," *Journal of Information Security*, vol. 5, no. 2, pp. 56–64, 2014.

[5] K. Griffin, S. Schneider, X. Hu, and T.-c. Chiueh, "Automatic generation of string signatures for malware detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2009, pp. 101–120.

[6] National Institute of Standards and Technology, "Secure hash standard (shs)," *FIPS PUB 180-4*, 2015.

[7] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Sur- veys*, vol. 44, no. 2, pp. 1–42, 2012.

[8] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," in *IEEE Symposium on Security and Privacy*. IEEE, 2007, pp. 32–39.

[9] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys*, vol. 50, no. 3, pp. 1–40, 2017.

[10] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining meth- ods for detection of new malicious executables," in *IEEE Symposium on Security and Privacy*. IEEE, 2001, pp. 38–49.

[11] V. M. Alvarez, "Yara: The pattern matching swiss knife for malware researchers," https://virustotal.github.io/yara/, 2024.

[12] A. Cohen, N. Nissim, and Y. Elovici, "Yara-based detection of malware," in *International Conference on Cyber Security Cryptography and Ma- chine Learning*. Springer, 2020, pp. 213–230.

[13] Cisco Systems, "Clamav: Open source antivirus engine," https://www. clamav.net/, 2024.

[14] VirusTotal, "Virustotal: Free online virus, malware and url scanner," https://www.virustotal.com/, 2024.

[15] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practi- tioner's Approach*, 8th ed. McGraw-Hill Education, 2014.

[16] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," in *Annual International Cryptology Conference*. Springer, 2017, pp. 570–596.

[17] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," *RFC 2104*, 1997.

[18] S. Ramirez, "Fastapi: Modern, fast web framework for building apis with python," https://fastapi.tiangolo.com/, 2024.
[19] Microsoft, "Typescript: Javascript with syntax for types," https://www.typescriptlang.org/, 2024.
[20] D. R. Hipp, "Sqlite: A self-contained, serverless sql database engine," https://www.sqlite.org/, 2024.
[21] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000, chapter 5: Representational State Transfer (REST).
[22] European Institute for Computer Antivirus Research, "Eicar standard anti-virus test file," 2003.