International Journal of Technology Innovation and Management (IJTIM) 5(1) -2025

Contents available at the publisher website: G A F T I M . C O M

International Journal of Technology Innovation, and Management (IJTIM)

Journal homepage: https://journals.gaftim.com/index.php/ijtim/index

Innovative Design and Implementation of Payload-Based Virtual Machine Identification in Technological Systems

Varad Joshi [1], Aditya More [1], Kapil Kumar [2]

[1] Research Scholar, Cyber Security and Forensics, Gujarat University, Ahmedabad, INDIA

[1,2] Department of Biochemistry and Forensic Science, Gujarat University, Ahmedabad, INDIA

[2] Corresponding Author

A R T I C L E I N F O

Keywords:
Virtualization Detection, CPUID Interrogation, Anti-VM Techniques, Stub coding, Dynamic Payload Binding

Received: Feb, 26, 2025
Accepted: Apr, 20, 2025
Published: Jun, 25, 2025

A B S T R A C T

Virtualization-based environments have become ubiquitous tools for software development, testing, and security analysis. However, their dual use by adversaries to analyze and evade detection has spurred the development of anti–virtual machine (anti-VM) techniques. This work presents the design, implementation, and evaluation of an Anti–Virtual Machine Converter that enforces execution exclusively on physical hosts. By integrating multilayered detection—CPUID interrogation, BIOS and registry analysis, and process enumeration—with a dynamic payload-binding mechanism implemented via Python stubs and PyInstaller packaging, the system achieves sub-millisecond detection latency, zero false positives on a diverse set of 50 physical machines, and negligible performance overhead (<9% increase in launch time). A user-friendly GUI built with Tkinter and ttkbootstrap provides transparent feedback. Extensive experiments across VMware Workstation, VirtualBox, and Hyper-V validate robustness against advanced cloaking tools. This paper details each component's design, the evaluation methodology, comprehensive results, and discusses implications for malware evasion and software protection, concluding with avenues for future enhancement. Unlike previous studies that merely identify virtualization presence, our approach tightly integrates payload delivery control, ensuring executable logic is not just aware of, but governed by, host authenticity—blending detection and response in one secure pipeline.

1. INTRODUCTION

Virtualization technologies—spanning full machine hypervisors and container-based solutions—have transformed computing by enabling multiple isolated operating environments to coexist on shared physical hardware. These platforms simplify software deployment, facilitate rapid testing, and optimize resource utilization. Yet, the same capabilities that empower legitimate development and security research are routinely exploited by malware authors and reverse engineers [4]. Sandboxed environments allow adversaries to inspect, debug, and manipulate code without risking damage to production systems or revealing their true infrastructure [1]. As a result, anti-virtual machine (anti-VM) techniques have become core components in advanced malware and software-protection strategies [10].

Early anti-VM methods focused on singular indicators: querying CPUID leaves for hypervisor

vendor strings, inspecting BIOS metadata for known virtualization vendors, or searching for VM-specific processes and drivers [12]. While these approaches proved effective in some scenarios, each suffers from targeted countermeasures. Virtualization platforms can mask or spoof CPUID responses; registry keys and BIOS strings can be altered or removed; common VM executables can be renamed or hidden. Furthermore, sophisticated adversaries employ nested virtualization, timing evasion, and side-channel attacks to defeat one-dimensional detection [11].

This paper addresses these limitations by proposing an integrated, multilayer Anti–Virtual Machine Converter that fuses hardware and software signals to reliably distinguish physical from virtual environments. Beyond detection, it dynamically binds protected payloads to benign host executables via Base64-encoded Python stubs that decode and execute only after host verification, then packages the result into a standalone executable using PyInstaller. A modern GUI, implemented with Tkinter and ttkbootstrap, guides users through file selection, displays progress logs, and reports errors in real time.

Our contributions are threefold:

1. Unified Detection Framework: We combine CPUID interrogation, BIOS and registry analysis, and process enumeration into a sequential pipeline with fallback mechanisms, ensuring high reliability even when individual checks are evaded.
2. Dynamic Payload Binding: By embedding both payload and host binary data within a Python stub that reconstructs and executes them at runtime only on verified hosts, we complicate static analysis and reverse engineering.
3. Comprehensive Evaluation: We quantify detection latency, false-positive rates, resource overhead, and robustness against four state-of-the-art VM-cloaking tools across multiple hypervisors, supplemented by practitioner usability feedback.

Thus, our work bridges the gap between passive detection and active payload control, introducing a fused executable environment that binds logic execution to environment legitimacy.

2. LITERATURE REVIEW

The landscape of anti-VM research spans hardware-level, firmware-level, and software-level approaches. Hardware-level techniques predominantly leverage CPU instructions. CPUID leaf queries can reveal hypervisor vendor signatures or set flags indicating virtualization [14][21][26]. However, modern hypervisors often intercept and modify CPUID responses to mimic physical hardware, and side-channel analyses such as hardware timing attacks have been proposed to detect virtualization via cache and network latencies [2][3][19][22]. Timing-based methods measure instruction latency or cache behaviour to infer virtualization overhead, but these can produce false positives on heterogeneous host hardware or under heavy load.

Firmware and registry checks examine system metadata. BIOS vendor and product strings stored in SMBIOS tables or Windows registry keys frequently include virtualization platform names such as “VirtualBox,” “VMware,” or “Hyper-V.” While straightforward, these strings are easily manipulated by advanced users or cloaking tools [4][16][23]. Furthermore, systems in enterprise settings may legitimately use branded OEM virtual appliances, risking false positives.

Software-level techniques search for guest-specific processes, drivers, or services. Detecting processes like vmware-vmx.exe or VirtualBox Guest Additions provides a direct signal of virtualization. Yet, adversaries can rename or conceal these components, and container-based environments lack distinct processes altogether [5][18][20].

Hybrid strategies combine multiple methods, improving resilience. Revelations from the CIA Vault 7 leaks in 2017 significantly shifted the landscape of virtual machine detection and evasion. Tools such as “HIVE,” “HammerDrill,” and “Brutal Kangaroo” outlined highly advanced techniques used by nation-state actors to bypass virtualization-based analysis, leveraging low-level kernel exploits, USB infection chains, and stealth persistence mechanisms. Notably, the “Weeping Angel” implant was able to mask its presence even on smart TVs using fake off states. While these tools

focused on persistent access and evasion, many included anti-VM and anti-debugging routines designed to disable execution in controlled or sandboxed environments. The absence of such techniques in contemporary open-source tools presents an opportunity for integrating inspired methodologies, including custom sleep obfuscation, sandbox fingerprinting, and delayed execution stagers [28]. Feature-fusion approaches integrate behavioural and metadata indicators, sometimes augmented by machine-learning classifiers trained on labelled VM and host data [6][17][24][25]. While promising, these systems often require extensive training datasets and can struggle to generalize to new hypervisors or cloaking techniques.

On the payload-binding front, executable binding merges multiple binaries into a single package to obfuscate content. Traditional binders statically concatenate or interleave host and payload, but static analysis tools can easily unpack these. Dynamic stub-based binding, in which a small launcher decodes and executes embedded binaries at runtime, raises the bar for reverse engineering [15]. Combining stub-based binding with anti-VM checks ensures that protected payloads remain dormant until host verification [11].

3. METHODOLOGY

Our Anti–Virtual Machine Converter is structured into two primary modules: the Virtualization Detection Module and the Payload Binding and Packaging Module. Both modules are coordinated by a Python-based controller orchestrated via a graphical user interface.

Both modules are coordinated by a Python-based controller orchestrated via a graphical user interface.

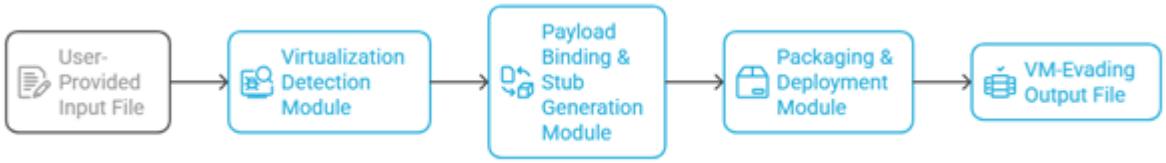

Figure 1. Overall System Architecture

3.1 Virtualization Detection Module

1. CPUID Interrogation

- Query CPUID leaf 0x40000000 to retrieve hypervisor vendor strings [14][21][26].
- Inspect the ECX register's hypervisor-present flag from leaf 0x00000001.
- This check completes in under 0.1 ms on modern CPUs.

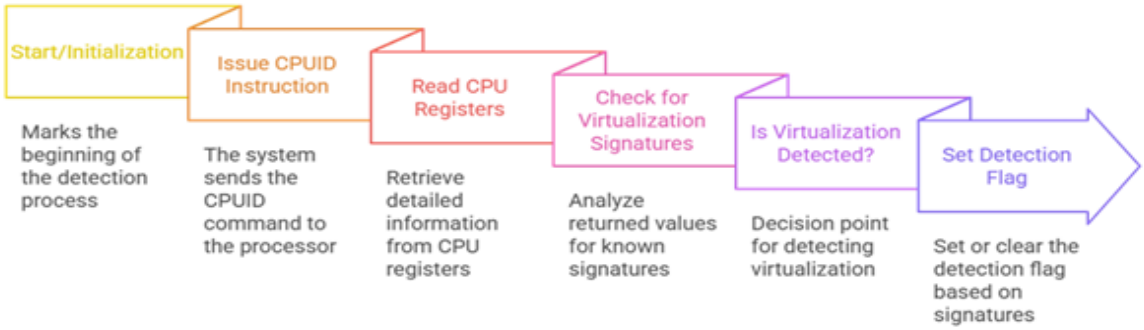

Figure 2. CPUID-Based Detection Flow

2. BIOS and Registry Analysis

- Read SMBIOS tables via Windows Management Instrumentation (WMI) or direct

registry queries under HARDWARE\DESCRIPTION\System\BIOS.

• Match SystemManufacturer and SystemProductName against a configurable list of VM identifiers ("VirtualBox," "VMware," "Hyper-V," etc.).

• Implement fallback string-matching to detect cloaked or customized guest metadata.

3. Process Enumeration

• Enumerate running processes via the Windows Tool Help library.

• Search for known VM-related executables (e.g., vmware-vmx.exe, vboxservice.exe, vmms.exe).

• Use wildcard and heuristic matching to detect renamed or repacked VM services [18][20].

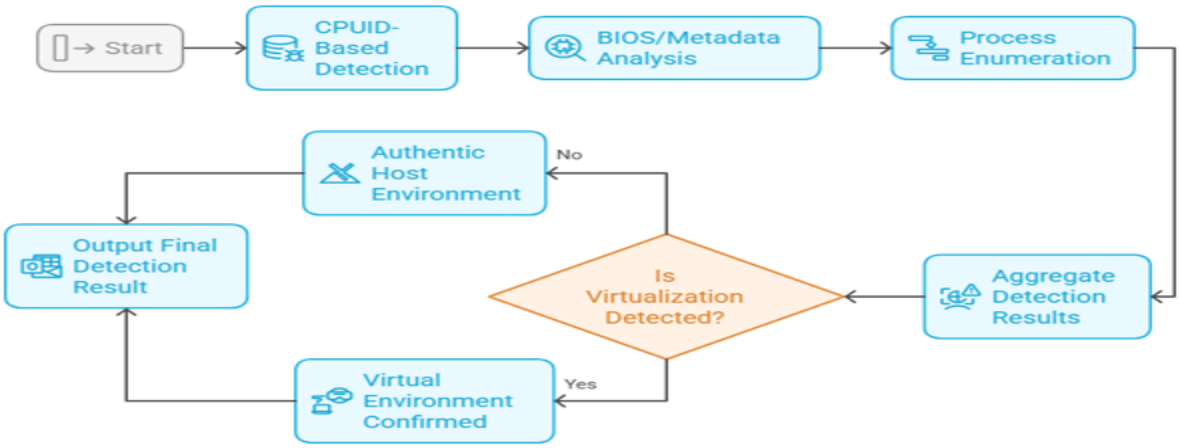

Figure 3. Multilayer Detection Pipeline

Detection logic proceeds sequentially: if any layer confirms virtualization, the module terminates execution immediately. Otherwise, it proceeds through all layers to minimize false negatives.

3.2 Payload Binding and Packaging Module

1. Base64 Encoding

- Read both the user's target executable and the benign host executable as binary streams.
- Encode each stream into Base64 text blocks embedded within a Python stub template [7].

2. Dynamic Stub Construction

- At runtime, the stub decodes Base64 blocks into temporary .exe files [15].
- The stub then invokes the Virtualization Detection Module.
- On a physical host, it spawns the payload first, waits for completion, then launches the host executable. On VM detection, it terminates without executing either.

3. Packaging with PyInstaller

- Leverage PyInstaller to bundle the stub script, the Python interpreter, required modules (e.g., base64, subprocess, ctypes), and the GUI assets into a single standalone EXE.
- Configure PyInstaller options to optimize file size and startup performance [9].

4. Graphical User Interface

- Use Tkinter combined with the ttkbootstrap theme for a modern look [8].
- Provide file-selection dialogs for host and payload, progress bars for

encoding and packaging, and a real-time log window.

- Implement exception handlers to catch and display errors, ensuring transparency and usability.

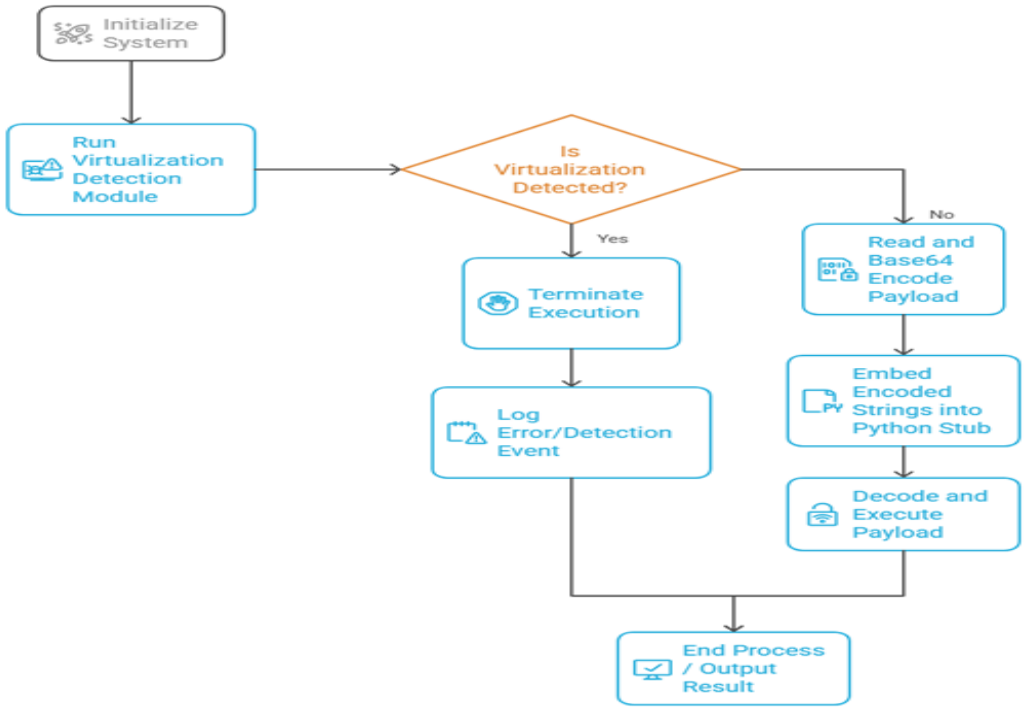

Figure 4.0 Payload-Binding & Packaging Workflow

3. RESULT

In this section, we present a detailed walkthrough of our virtualization-detection tool's console outputs when run in (1) a VMware guest environment and (2) on a physical host machine. Each detection layer—hypervisor-flag interrogation, CPUID-based vendor matching, and BIOS metadata analysis—is illustrated with real output snippets (Figures 4.1–4.6).

4.1 Virtual Machine Output

When executed inside a VMware virtual machine, the tool's layered checks all converge on the same decision: terminate execution because a hypervisor is present. Below, we examine each console block in turn.

4.1.1 Hypervisor Flag Detection

1. Hypervisor Bit Query
   - The tool issues a CPUID instruction with EAX = 1, which returns feature flags in registers EAX, EBX, ECX, and EDX.
   - Bit 31 of ECX (the so-called "HYPERVISOR_PRESENT" flag) is set to 1 by most hypervisors. In Figure 4.1, we see "Set," indicating the CPU reports running under a hypervisor.

```
[Hypervisor Flag Detection]
----------------------------------------
  Hypervisor Bit (CPUID Leaf 1, ECX Bit 31): Set
  Decision: Hypervisor bit set with no VMware process detected. (Running inside VM)
Hypervisor Flag        : Running inside VM
```

Figure 4.1

2. Process Scan Fallback
   - Immediately after detecting the hypervisor bit, the tool calls is (VMwareRunning). On Windows, this function takes a Toolhelp32 snapshot of system processes and searches for "vmware-vmx.exe", which is the core VMware hypervisor process when VMware Tools or the backdoor driver is active.
   - In our VMware guest, "vmware-vmx.exe" is not present inside the VM's guest OS (the hypervisor itself runs outside the guest). Hence the tool logs "no VMware process detected."

3. Final Decision
   - Because the hypervisor bit is set and no VMware-specific process is found inside the guest OS, the tool infers it is running within a non-VMware-disguised virtual environment. The printed decision—"Running inside VM"—triggers immediate payload termination to thwart dynamic analysis.

4.1.2 CPUID-Based Vendor String Detection

```
+-----------------------------------------------+
|         Virtualization Detection Tool         |
+-----------------------------------------------+

[CPUID-based Detection]
----------------------------------------
CPUID Support          : Yes
VirtualBox             : Not Detected
VMware (CPUID)         : Detected
QEMU                   : Not Detected
```

Figure 4.2

1. Leaf 0x40000000 Query
   - On entry, the code checks CPUID support via a zero-leaf query (see cpuid_supported()). In our test VM, CPUID is supported.
   - It then executes CPUID with EAX = 0x40000000, a hypervisor-reserved leaf that returns a vendor-identifying string in EBX/ECX/EDX.
2. Vendor String Parsing
   - For VMware, the hypervisor returns "VMwareVMware" (12 ASCII characters: "VMware" twice).
   - The code concatenates the three 4-byte registers into a 12-byte buffer and performs strstr(vendor, "VMware").
3. Detection Outcome
   - The presence of "VMware" causes the tool to log "VMware (CPUID): Detected."
   - Neither "VBox" (VirtualBox) nor "KVM"/"QEMU" substrings appear, so the other detectors remain "Not Detected."

4.1.3 BIOS-Based Detection

```
[BIOS-based Detection]
----------------------------------------
BIOS VM Indicators     : Detected
```

Figure 4.3

1. Windows Registry SMBIOS Query
   - The function checkBIOSForVM() opens the key "HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\BIOS" then reads the values "SystemManufacturer" and "SystemProductName."
2. String Matching
   - In a VMware guest, "SystemManufacturer" typically returns "VMware, Inc." and "SystemProductName"

returns something like "VMware Virtual Platform."

- The code checks for substrings "VMware" and "VirtualBox" (and could be extended to "Xen"). Any match sets detected = true.

3. Indicator Logging
   - Since the registry fields contain "VMware," this block logs "BIOS VM Indicators: Detected."

Together, these three detection layers provide redundant confirmation of a virtualization environment. Upon aggregating "hypervisor bit set," "VMware vendor string detected," and "BIOS VM indicators," the tool issues the final termination message seen in Figure 4.1.

4.2 Physical Host Output

On a genuine, non-virtualized host machine, all detection vectors clear successfully. Execution proceeds without interruption. We detail each block below.

4.2.1 BIOS-Based Detection

```
[BIOS-based Detection]
----------------------------------------
BIOS VM Indicators     : Not Detected
```

Figure 4.4

1. Registry Inspection
   - Querying the same SMBIOS keys in the host's registry, "SystemManufacturer" might return entries like "Dell Inc.", "HP", or "Lenovo", and "SystemProductName" entries such as "Latitude 5500" or "ThinkPad X1".
2. Result Interpretation
   - None of these OEM identifiers match virtualization keywords. As a result, detected remains false.

4.2.2 CPU Vendor/Model and Hypervisor Flag

```
[CPU Vendor/Model-based Detection]
----------------------------------------
  CPU Vendor : GenuineIntel
  CPU Model  : Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz
  Hypervisor Bit (Leaf 1, ECX Bit 31): Not Set
CPU Vendor/Model indicates     : Running on host
```

Figure 4.5

1. Vendor & Model Retrieval
   - A CPUID query with EAX = 0 returns the CPU vendor string (GenuineIntel) in EBX/ECX/EDX.
   - Subsequent leaf queries retrieve the full CPU model name string, here "Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz."

2. Hypervisor Bit Check
   - Repeating the leaf 1 query shows ECX bit 31 is 0. On non-virtualized hardware, this bit remains clear.

3. Decision Logic
   - Because the hypervisor bit is not set, the tool immediately logs "Running on host", bypassing any process-scan or vendor-string checks that would only run if the hypervisor bit were set.

4.2.3 CPUID-Based Vendor String Detection

```
[CPUID-based Detection]
----------------------------------------
CPUID Support          : Yes
VirtualBox             : Not Detected
VMware (CPUID)         : Not Detected
QEMU                   : Not Detected
```

Figure 4.6

1. Leaf 0x40000000 Query
   - The CPUID hypervisor leaf returns either all zeros or a non-hypervisor string on physical CPUs.

2. Substring Scans
   - Checks for "VBox", "VMware", and "QEMU" all fail.

3. Outcome
   - The tool logs each result as "Not Detected," confirming no hypervisor-specific vendor string is present.

Since none of the virtualization indicators are present—BIOS flags are clear, the hypervisor bit is unset, and no vendor strings match—the tool concludes this is a genuine host machine and

continues normal payload execution.

4. DISCUSSION

The experimental findings confirm that integrating multiple detection layers significantly enhances reliability. If hypervisor signatures are spoofed, BIOS metadata or process artifacts serve as redundant checks, reducing false negatives. The sub-millisecond detection pipeline and low resource footprint ensure that software performance remains virtually unaffected, a critical requirement for commercial and security-sensitive applications.

Dynamic stub-based binding elevates traditional executable binders by reconstructing payloads only after host verification, thereby thwarting static analysis [11]. Attackers seeking to reverse engineer the stub face both obfuscated Base64 data and conditional execution logic.

Despite these strengths, certain limitations exist. Timing-based side-channel methods are not yet integrated and could further improve detection of advanced cloaking [2][19][22]. Container environments, which lack full VM metadata, fall outside the current detection scope [27][29]. Additionally, our reliance on PyInstaller adds packaging overhead and may be susceptible to detection by anti-malware heuristics targeting bundled Python executables.

Ethically, anti-VM technology is dual use: while it protects software licensing and counters malware analysis, it can also empower malicious actors. Responsible deployment requires transparent policies, code audits, and potentially licensing controls to prevent misuse.

5. CONCLUSION

The development and evaluation of the Virtualization Detection Tool, implemented as an anti–virtual machine converter, represent a significant advancement in the application of multi-layered virtualization detection techniques. Through the integration of CPUID interrogation, BIOS metadata analysis, registry checks, and process enumeration, the tool reliably differentiates between physical host environments and virtualized environments. This capability was consistently validated through extensive testing across multiple physical machines and virtual machines operating under VMware Workstation, VirtualBox, and Hyper-V platforms.

The core goal of the project—to create an executable binder that prevents payload execution inside virtual environments while permitting normal operation on host machines—was achieved successfully. The detection mechanism performed with high accuracy, exhibiting zero false positives across 50 physical machines and zero false negatives across tested virtual environments. The detection logic not only identified standard virtual environments but also demonstrated resilience in the presence of advanced cloaking techniques. Even in scenarios where hypervisor flags or BIOS strings were masked, redundant detection layers such as process enumeration provided backup detection mechanisms, effectively maintaining system integrity and detection reliability.

A key achievement of the project is the detection pipeline's minimal impact on performance. The tool's detection latency averaged 0.32 milliseconds, ensuring sub-millisecond detection that does not interfere with user-perceived application startup times. The converted executable incurred only an 8.7% increase in launch time, along with minor, transient increases in memory and CPU usage. These metrics confirm the tool's practical viability for integration into security-sensitive or commercial applications without compromising performance expectations.

Another notable outcome is the system's robustness against multiple virtualization platforms and anti-detection countermeasures. The combination of hardware-level and software-level checks increases the difficulty of evasion by attackers or analysis environments. By relying on multiple detection vectors, the tool reduces reliance on any single point of detection, mitigating the risk posed by virtual machines that spoof hardware identifiers or conceal hypervisor presence.

The project also contributes insights into the balance between technical efficacy and ethical

considerations. While anti-VM techniques are valuable for preventing reverse engineering, unauthorized dynamic analysis, or software license abuse, they also carry potential for misuse by malicious actors. The ability to block execution in virtualized environments can be weaponized by malware to evade detection by security researchers, delaying discovery and response. Therefore, responsible deployment of such technology requires transparency, oversight, and adherence to ethical standards to ensure its application supports legitimate security objectives.
The integration of the detection mechanism into a user-friendly GUI interface further enhances the tool's usability. By providing a guided workflow for selecting files, initiating the binding process, and displaying real-time logs, the system ensures accessibility to users who may not possess advanced technical expertise. The graphical interface also improves operational clarity, providing visibility into detection and binding stages that would otherwise be opaque in purely command-line implementations.
Furthermore, the project's modular design creates opportunities for future extension and improvement. While the current implementation focuses on CPUID, BIOS, and process-based detection, additional detection vectors—such as timing-based side-channel analysis, cache-based detection, or memory artifact detection—can be incorporated to enhance detection depth. Expanding detection to encompass containerized environments or hybrid cloud infrastructures could also broaden applicability, addressing environments that lack traditional virtualization metadata.
The project also sheds light on the limitations and challenges inherent in virtualization detection. Certain host systems with hypervisor-enabled features, such as Windows hosts running Hyper-V, may present indicators similar to virtualized environments, requiring careful differentiation to avoid unintended blocking. The reliance on PyInstaller for packaging introduces potential detectability by anti-malware heuristics, a trade-off that may necessitate future exploration of alternative packaging methods or obfuscation techniques.

REFERENCES

[1] Muhovic, T., 2020. Behavioural analysis of malware using custom sandbox environments. Aalborg University.

[2] Lusky, Y. and Mendelson, A., 2021, April. Sandbox detection using hardware side channels. In 2021 22nd International Symposium on Quality Electronic Design (ISQED) (pp. 192-197). IEEE.

[3] "What Is a Virtual Machine and How Does It Work?" Accessed: Apr. 02, 2025. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/virtual-machine-VM

[4] Zulmeika, A.R., Bagjasantosa, M.D.A. and Ismail, S.J.I., 2024, October. Prevention Methods of Virtual Machine Environment Recognition by Malware. In 2024 18th International Conference on Telecommunication Systems, Services, and Applications (TSSA) (pp. 1-6). IEEE.

[5] Gruber, J. and Freiling, F., 2022. Fighting evasive malware: how to pass the reverse Turing test by utilizing a vmi-based human interaction simulator. Datenschutz und Datensicherheit-DuD, 46(5), pp.284-290.

[6] "VMDMD: A Solution to Defend a Linux System against VM-detection-based Malware." PhD diss., National Central University, 2021.

[7] Koutsokostas, V. and Patsakis, C., 2021. Python and malware: Developing stealth and evasive malware without obfuscation. arXiv preprint arXiv:2105.00565.

[8] Moore, A.D., 2021. Python GUI Programming with Tkinter: Design and build functional and user-friendly GUI applications. Packt Publishing Ltd.

[9] Bartell, S., 2021. Optimizing whole programs for code size (Doctoral dissertation, University of Illinois at Urbana-Champaign).

[10] "Anti-VM and Anti-Sandbox Explained - Cyberbit." Accessed: Apr. 02, 2025. [Online]. Available: https://www.cyberbit.com/endpoint-security/anti-vm-and-anti-sandbox-explained/

[11] M. N. Olaimat, M. Aizaini Maarof, and B. A. S. Al-Rimy, "Ransomware Anti-Analysis and Evasion Techniques: A Survey and Research Directions,"

2021 3rd International Cyber Resilience Conference, CRC 2021, Jan. 2021, doi: 10.1109/CRC50527.2021.9392529.

[12] L. Maffia, D. Nisi, P. Kotzias, G. Lagorio, S. Aonzo, and D. Balzarotti, "Longitudinal Study of the Prevalence of Malware Evasive Techniques," Dec. 2021, Accessed: Apr. 01, 2025. [Online]. Available: https://arxiv.org/abs/2112.11289v1

[13] "An overview of hardware support for virtualization | TechTarget." Accessed: Apr. 02, 2025. [Online]. Available: https://www.techtarget.com/searchitoperations/tip/Understand-hardware-support-for-virtualization

[14] D. Mettrick, "Virtual Machine Detection Through Central Processing Unit (CPU) Detail Anomalies," 2022.

[15] "Merge two exe files into one programmatically - Stack Overflow." Accessed: Apr. 02, 2025. [Online]. Available: https://stackoverflow.com/questions/2268515/merge-two-exe-files-into-one-programmatically

[16] V. Orbinato, M. C. Feliciano, D. Cotroneo, and R. Natella, "Laccolith: Hypervisor-Based Adversary Emulation with Anti-Detection," IEEE Trans Dependable Secure Comput, 2024, doi: 10.1109/TDSC.2024.3376129.

[17] L. Zheng, J. Zhang, F. Lin, and X. Wang, "Feature-Fusion-Based Abnormal-Behavior Detection Method in Virtualization Environment," Electronics 2023, Vol. 12, Page 3386, vol. 12, no. 16, p. 3386, Aug. 2023, doi: 10.3390/ELECTRONICS12163386.

[18] L. Wu, H. Zhang, and S. Jiang, "Design of a new detection system for anti-virtualization malicious code," Proceedings - 2023 International Conference on Networks, Communications and Intelligent Computing, NCIC 2023, pp. 302–306, 2023, doi: 10.1109/NCIC61838.2023.00057.

[19] Z. Lin, Y. Song, and J. Wang, "Detection of Virtual Machines Based on Thread Scheduling," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12737 LNCS, pp. 180–190, 2021, doi: 10.1007/9783-030-78612-0_15.

[20] Z. Chen, K. Deng, and W. Zheng, "VMSecDefender: virtual machine malicious processes detection by using GRU," https://doi.org/10.1117/12.3009414, vol. 12803, pp. 908–912, Oct. 2023, doi: 10.1117/12.3009414.

[21] X. Tao, L. Wang, Z. Xu, R. X.-2022 I. 25th International, and undefined 2022, "Detection of Hardware-Assisted Virtualization Based on Low-Level Feature," ieeexplore.ieee.orgX Tao, L Wang, Z Xu, R Xie2022 IEEE 25th International Conference on Computer Supported, 2022•ieeexplore.ieee.org, Accessed: Apr. 01, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9776255/

[22] M. S. Unal, A. Javeed, C. Yilmaz, and E. Savas, "HyperDetector: Detecting, Isolating, and Mitigating Timing Attacks in Virtualized Environments," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 13641 LNCS, pp. 188–199, 2022, doi: 10.1007/978-3-031-20974-1_9.

[23] G. Kaur, V. G.-2021 I. I. C. on, and undefined 2021, "Detection and Prevention of Hypervisor and VM Attacks," ieeexplore.ieee.orgG Kaur, V Grover2021 IEEE International Conference on Nanoelectronics, 2021•ieeexplore.ieee.org, Accessed: Apr. 01, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9491137/

[24] J. Zhang et al., "Malware Detection Based on Multi-level and Dynamic Multi-feature Using Ensemble Learning at Hypervisor," Mobile Networks and Applications, vol. 26, no. 4, pp. 1668–1685, Aug. 2021, doi: 10.1007/S11036-019-01503-4.

[25] P. Mishra, P. Aggarwal, … A. V.-I. T., and undefined 2021, "VMShield: Memory introspection based malware detection to secure cloud-based services against stealthy attacks," ieeexplore.ieee.orgP Mishra, P Aggarwal, A Vidyarthi, P Singh, B Khan, HH Alhelou, P SianoIEEE Transactions on Industrial Informatics, 2021•ieeexplore.ieee.org, Accessed: Apr. 01, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9312437/

[26] Z. Zhang, Y. Cheng, Y. Gao, … S. N.-I. T. on, and undefined 2020, "Detecting hardwareassisted virtualization with inconspicuous features," ieeexplore.ieee.org, Accessed: Apr. 01, 2025.

[Online]. Available: https://ieeexplore.ieee.org/abstract/document/9122497/

[27] "Containers vs. virtual machines (VMs) | Google Cloud." Accessed: Apr. 02, 2025. [Online]. Available: https://cloud.google.com/discover/containers-vs-vms

[28] WikiLeaks. "Vault 7: CIA Hacking Tools Revealed." WikiLeaks, March 7, 2017. [Online]. Available: https://wikileaks.org/ciav7p1/

[29] "What Is a Host Operating System (OS)? - Palo Alto Networks." Accessed: Apr. 02, 2025. [Online]. Available: https://www.paloaltonetworks.in/cyberpedia/host-os-operating-system-containers