MAYAVI: A Cyber-Deception Hardware for Memory Load-Stores

Preet Derasari
The George Washington University
Washington, D.C., USA
preet_derasari@gwu.edu

Kailash Gogineni
The George Washington University
Washington, D.C., USA
kailashg26@gwu.edu

Guru Venkataramani
The George Washington University
Washington, D.C., USA
guruv@gwu.edu

ABSTRACT

Rapid evolution of security attacks presents a perpetual challenge to computer system defenders in terms of continuously upgrading their defense capabilities and being aware of adversarial tactics. Emerging technologies like *cyber-deception* offer the unique advantage of intelligently surveying hostile behavior while actively safeguarding sensitive assets by manipulating the malware execution flow to non-useful states or misrepresenting critical data.

This paper explores the untapped potential of a hardware-assisted cyber-deception framework that augments a contemporary processor's abilities to realize a proactive deception platform. We present MAYAVI, a load/store unit-based hardware deception engine that dynamically alters the target addresses of memory requests issued by a malicious process. The *redirected* requests can lead to honeypots that actively engage and deceive the adversary. Our experimental results show MAYAVI's efficacy against recent malware families while incurring negligible performance impact.

CCS CONCEPTS

• **Security and privacy** → **Hardware security implementation**;
• **Computer systems organization** → *Architectures*.

KEYWORDS

Cyber-deception, Hardware security, Proactive defense

ACM Reference Format:
Preet Derasari, Kailash Gogineni, and Guru Venkataramani. 2023. MAYAVI: A Cyber-Deception Hardware for Memory Load-Stores. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590272

1 INTRODUCTION

The current trend in cyber-security threats has shown a rapid rise in attacks with resiliency against conventional defense implementations, the majority of which rely on detect-and-defend [6, 21] or system security-hardening techniques [8]. In most cases, inflexible approaches limit the effectiveness of such strategies, especially against never-before-seen threats [4]. Ultimately, securing computer systems against attackers becomes a continuous cycle of creating defenses that seek to patch emerging vulnerabilities. Breaking this pattern calls for investing in *proactive defense* strategies to tackle the evolving nature of cyber threats.

To this end, *cyber-deception* has shown promise as a proactive defense paradigm that intentionally allows an adversary to interact with the system and learn about their behavior by *luring* them into believing that they are operating successfully [2]. This can provide rich information for threat intelligence, decrease the attacker footprint by proactively catching them, and create forensics to formulate better defense procedures during cyber attacks. Cyber-deception has rapidly grown as a next-generation security solution even in zero-trust architectures and mission-critical systems [14].

Prior cyber-deception frameworks primarily relied on software-based approaches and required non-trivial modifications to software and system implementations. Unfortunately, these can create noticeable perturbations to the malware's runtime execution profiles, providing plenty of opportunities for sophisticated attackers to become aware of the defender's presence, which may ultimately defeat the purpose of a cyber-deception framework. Recent work shows how attackers can scan a system's wear-and-tear artifacts [24] and discover the existence of fake execution platforms like virtual machine (VM) honeypots in deception-based networks [26].

In pursuit of effective cyber-deception frameworks that can remain transparent to attackers, we note that hardware support offers certain unique advantages. 1. Hardware can deploy deception *dynamically*, reducing the lead time made available to the adversaries to react and re-calibrate their offenses. 2. Hardware-based deception results in a near-native execution profile for the executing malware, thus causing a low-performance impact and avoiding any noticeable changes to the system environment that may provoke an attacker.

This paper explores hardware-based cyber-deception architecture that capitalizes on the hardware's ability to provide an effective defense. Note that most of the malware, including ransomware [17] and InfoStealers [30], are reliant upon memory accesses to accomplish their attacks, providing a rich opportunity for hardware to transparently (invisible) modify the memory requests and preserve the integrity of sensitive system assets. We present MAYAVI, a load/store unit deception engine that dynamically replaces the target addresses of specific memory requests issued by the processor on behalf of the malicious code toward accessing sensitive memory locations. The *redirected* memory requests may actively mislead the adversary with either *honey* addresses containing plausible-looking, counterfeit information to engage the attacker continually or *randomly* selected addresses to confuse the malware with irrelevant data.

We evaluate MAYAVI's effectiveness and runtime impact using five samples from prominent malware families that encrypt sensitive files (ransomware), exfiltrate user information (InfoStealer), or

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590272

manipulate pointers to execute arbitrary code (buffer overflow). Our case studies show that MAYAVI can successfully deflect attacker-emitted memory requests away from critical system resources. Moreover, the low execution time impact of our hardware deception operations makes it more challenging for the attackers to find any perceptible differences in the malware's execution time profile, thereby increasing the efficacy of cyber-deception.
In summary, the key contributions of this paper are as follows:

- We explore hardware support to efficiently implement cyber-deception to transparently counter the malware with a near-native execution profile, thereby making deception-related operations invisible to the attackers. To the best of our knowledge, this is the *first* work demonstrating hardware primitives for cyber-deception.
- We present MAYAVI, a *hardware deception engine design* demonstrating how to dynamically modify select memory requests issued by a malware process and redirect them away from damaging sensitive system assets.
- We evaluate MAYAVI against five malware families and show that hardware-supported cyber-deception can effectively deceive sensitive memory requests and do so with low-performance overheads (< 7%). This can improve the efficacy of cyber-deception.

2 THREAT MODEL AND ASSUMPTIONS

Our threat model considers attacks that are reliant on memory-related operations. Microsoft reports that over 70% of security vulnerabilities are exposed through memory bugs exploited by malware [23]. Notable examples of these attack classes include manipulating the file systems through directory mapping APIs [25], manipulating the memory vulnerabilities in legacy programs to hijack legit control flows, and exfiltrating sensitive user information [11]. Some malware most commonly employed in such attack types are ransomware, spyware (InfoStealers), and trojans that sneak through buffer overflow vulnerabilities.

State-of-the-art techniques that use DNN-based malware detectors [27] have reported over 97% accuracy in detecting malicious binaries at program load time. We assume that a victim system is equipped with similar malware detectors to classify an unknown application's intentions (malicious vs. benign) with reasonably high accuracy. Once classified as malware, a system administrator can either naively purge the binary from the system (lost opportunity to learn about the malware's dynamic behavior) or allow it to execute on a cyber-deception framework assisted by our MAYAVI engine. The latter approach not only protects the system from damage but also enlightens the defenders about the attacker's intent using the profile data collected during its runtime for more robust defense designs in the future.

3 MAYAVI DESIGN

In this section, we describe the fundamental components of our hardware design and show how they work together to achieve runtime deception against malware.

One of the critical requirements for effective deception is to minimize the possibility of being detected by the adversary for the longest time. As such, the hardware support for achieving this goal should be lightweight and integrated into the processor pipeline to avoid leaving forensics for adversaries to observe physically (e.g., timing differences) or audit (e.g., increased memory traffic/bandwidth).

We start by exploring the hardware design possibilities to efficiently (and transparently) edit the memory requests issued to sensitive addresses by the malware. Our proposed hardware *Deception Engine* integrates tightly with the operations of an out-of-order processor's load/store (LD/ST) unit such that any processor-outgoing memory request targeting sensitive address ranges will be deceived via address replacement or obfuscation before being sent to the cache hierarchy. Figure 1 shows our MAYAVI hardware design.

3.1 Range Filter

MAYAVI incorporates an address-checking mechanism to precisely target the deception actions on memory requests involving sensitive information (succinctly called *sensitive memory requests*). Note that sensitive memory locations can be discovered using pre-runtime binary analysis tools, such as angr [29].

During malware execution, the target addresses (*Mem Req Addr*) of every sensitive memory request waiting to be issued by the processor inside the load/store queue are intercepted by the *range filter* mechanism. Our design leverages efficient address-watch mechanisms [31], where each target memory location is compared against multiple address ranges stored within the range filter. The endpoint addresses are stored in two sets of hardware registers, i.e., *lower bound address (LBA)* and *upper bound address (UBA)*. The LBA and UBA values are initialized by the pre-runtime analysis tool [29].

Using a simple comparator logic (*Cmp*), the range filter checks the target address of an incoming memory request against each unique address range, *(LBA1, UBA1) ... (LBAn, UBAn)*. The comparison results are stored bit-wise in an accumulator register where a '1' indicates a hit for the corresponding address range. For every sensitive memory request (*Mem Req*) that is a hit in the range filter, it enables the *Mode Lookup Unit*. Subsequently, at the end of the deception operation, the range filter signals a multiplexer to forward the modified memory request instead of the original *Mem Req* to the cache hierarchy.

3.2 Mode Lookup Unit

When deception is enabled for a memory (load/store) instruction by the range filter upon a match, the *mode lookup unit* receives the type of memory request (load or store) and the corresponding address range indicated by the position of '1' within the accumulator. Using this information, the mode lookup unit consults the lookup table to find the relevant mode of deception for the type of memory request accessing the matching address range.

We design and incorporate two deception modes and the hardware structures that implement them:

- *Honey Mode:* Replace the target addresses with honey addresses, which lures the attacker away from the sensitive data to a honey (fake) location in the relevant honeypot (e.g., filepath strings, dummy buffers).
- *Random Mode:* Replace the target address with a randomly selected address within a pre-specified window. The address

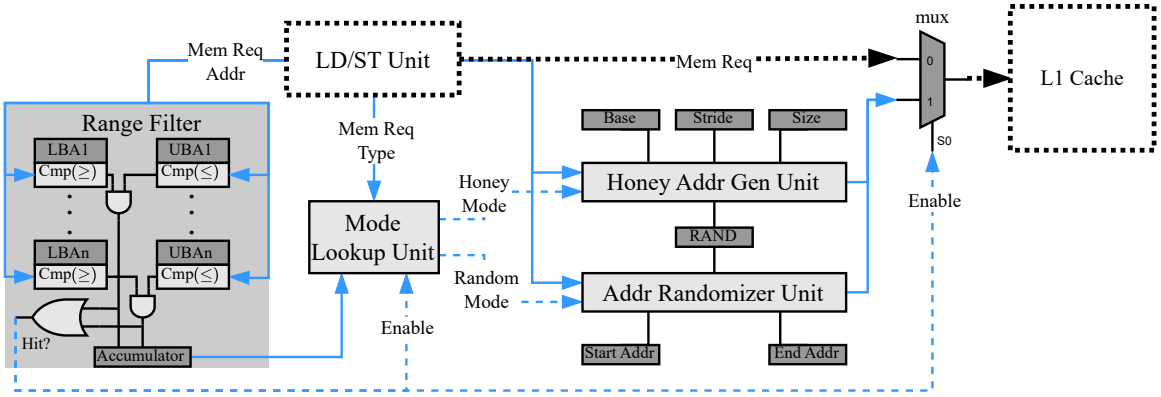

Figure 1: MAYAVI Deception Engine incorporating hardware support in LD/ST unit for memory address redirection and obfuscation functions. Deception-related hardware is highlighted using gray color.

window may contain random data that the attacker may deem useless.

The mode may be randomly chosen to prevent the adversaries from predicting MAYAVI's actions, thereby adding to the adversary's confusion. In our current design, for all of the watched sensitive address ranges discovered via binary analysis before program execution, the system administrator initializes the deception modes using privileged APIs. We note that the lookup table initializing process can be automated using deception strategy generators against specific malware types explored by software-based automation frameworks like SODA [28].

3.3 Honey Address Generation Unit

For each memory request to be deceived in *honey mode*, the *honey address generation unit* is enabled by the mode lookup unit to replace the original target addresses with a honey address containing pre-populated honeypot values. These honeypot values are populated by system administrators using address/data scrambling utility tools for susceptible data structures generally stored in the memory during program execution. For example, honeypots can contain honey files, pointers to honey code, and plausible-looking network parameters. These honeypots are loaded into the process memory alongside the malware, and their (honey) addresses are available to the hardware at runtime. To redirect sensitive memory requests, MAYAVI picks a honey address as follows:

$$honey\_addr = base + stride * (RAND \bmod (size/stride))$$

The *base*, *stride*, and *size* of a honeypot in memory denote the honeypot's base address, the length of each honeypot entry, and the total honeypot size (in bytes), respectively, and are stored in three reserved registers as shown in Figure 1. Modern processors (e.g., Intel Ivy Bridge) support random number generators in hardware [9], which may be used to calculate the RAND register's value. Using this deception mode, load requests can be redirected to specific honey resources, preventing attackers from hitting their target. For store operations, the attacker's data may be tracked by redirecting their requests to honey addresses that are actively monitored/logged by system defenders. For example, suppose an attacker wants to read a (sensitive) filepath string from a watched address region. In that case, the target address of all memory requests that load the relevant bytes will be replaced by a randomly chosen dummy filepath's address (honey_addr) within the honeypot memory location.

3.4 Address Randomizer Unit

For each memory request to be deceived in *random mode*, the *address randomizer unit* generates a random address value within a pre-specified address window whose start and end address values are stored in two reserved registers. The random fill cache architecture [20] shows how a missing cache line can be filled with a randomly selected neighboring address of the target. Our deception engine builds on this obfuscation support to pick a random address within a memory range initialized with arbitrarily generated data values at malware load-time. The randomizer unit replaces the target addresses of memory requests with the randomly generated memory address calculated as:

$$rand\_addr = start\_addr + RAND \bmod (end\_addr - start\_addr + 1)$$

start_addr and *end_addr* are reserved registers indicating the address range in memory containing the arbitrary data values. The RAND register acts as a seed to compute a random address between the start and end addresses. For all load requests, the address window specifies a memory region that may contain *garbage* data values and does not reveal any useful information to the attacker. For example, suppose that InfoStealer malware tries to read sensitive information like passwords. Using our deception engine, we may supply random garbage data values to the malware by randomizing the target addresses of all load requests accessing the data structure that holds the passwords in memory. In the case of store requests, the randomizer unit may be set up to calculate a *rand_addr* value inside a *sink* address range that is flushed periodically to prevent an attacker from making any changes to sensitive memory locations. For example, attempts made by ransomware to write encrypted contents to a user file may be diverted to random

addresses and flushed after a certain number of write attempts, thus preserving file integrity.

4 EXPERIMENTAL EVALUATION

We use the x86 build of Gem5 version 22.0.0.2 [5] as our evaluation platform. The MAYAVI LD/ST Unit Deception Engine is built on top of the existing Gem5 DerivO3 CPU model. All simulations are single process context threads execution of Gem5 configured with a 2GHz core frequency and 8GB of DDR4 memory.

4.1 Effectiveness of MAYAVI

To evaluate the effectiveness of the MAYAVI deception engine in assisting a cyber-deception framework, it is first important to understand the intrinsic behavior of various malware samples. With speed and stealth being the malware's goals to evade being caught by security defenders, malware typically iterates over a little code snippet to accomplish their attacks. Consequently, a finite number of (static) address ranges were sufficient to be monitored during malware runtime. We selected five malware samples representing some of the most potent attack vectors: three from the ransomware family, one buffer overflow benchmark, and one spyware sample. We discuss each malware and its attack chain, the maximum number of sensitive address ranges (in the static code) to be monitored obtained using code analyzers (e.g., angr [29]), the deception operations executed on watched memory addresses, and the post-deception outcome. For our evaluation, all malware binaries were statically compiled from their source codes, and any address space randomization tactics employed by the OS were turned off to keep the address ranges consistent over multiple runs. We note that in real scenarios, a binary analyzer [29] can discover the address ranges of interest regardless of the binary's nature. Table 1 presents a summary of our experimental results.

WannaCrypt0r [25] is a part of the WannaCry ransomware family that invades legacy computers and encrypts specific files. Its evaluation sample targeted a pre-defined list of directories and then opened, read, and overwrote all containing files using ciphertext. From our analysis, we discovered five sensitive address ranges (in the static code) that included the memory addresses of data structures for storing the *directory path* string, *file path* string, *renamed file path* string, the *read buffer*, and *write buffer* that store the file contents. MAYAVI enabled in *Honey Mode*, targeted the memory addresses of the *directory path* buffer storing discrete paths from a list of directories. Each memory request emitted by the malware accessing the buffer was deceived by replacing the target addresses with the honey directory path addresses supplied by the *honey address generator unit*. We observed that deception was triggered ≈ 567 times by MAYAVI to redirect all memory requests attempting to access 20 user directories. As a result of our deception, sensitive directories were protected from this malware.

The ***Bad Rabbit*** [22] ransomware family targets sensitive data inside vulnerable corporate networks. It starts traversal from a target directory and creates a list of all successive file paths. Then, for all file paths, it opens, reads, and overwrites them with encrypted contents and finally renames the files with a custom extension. According to our analysis of this sample, we discovered six sensitive address ranges, including the *directory*, *file*, and *renamed file* path strings, along with the *stat structure*, and the *read* and *write buffers*. MAYAVI targeted all memory requests trying to access the *file path* buffer in *Honey Mode* and replaced their target addresses with the honey files' locations. Our observations indicated that deception was triggered ≈ 1300 times to deceive all such memory requests. The security outcome for this sample was complete protection of all target files from the attacker's purview.

The ***Petya*** [1] ransomware family targets a victim's Master Boot Record (MBR) and encrypts all files. It generates a 256-bit key and sends it to a Command and Control (C&C) server using the IPv4 protocol. Each unencrypted file in the target directory is duplicated with a custom extension. The original files are deleted after their contents are encrypted (symmetric encryption) and stored in the ransomware-created duplicate files. This sample revealed six sensitive memory address ranges as suitable targets for deception, including the *directory* and *file* path strings, the *stat structure* used to determine the validity of each file, the *read* and *write* buffers storing the file's contents, and the *encryption key* buffer that eventually transmits to the remote attacker. MAYAVI's deception targets were the file and directory path string buffers. We observed that the deception engine was triggered ≈ 2520 times in *Honey Mode* to deceive all memory requests accessing the directory and file path buffers by replacing the target addresses with the honey file and directory path locations in memory. Eventually, the deception operations redirected the ransomware sample away from the user files and directories, thus preserving their integrity.

The ***RIPE*** [32] buffer overflow benchmark provides various attack configurations that present a standard platform to test the capabilities of a buffer overflow protection mechanism. In this benchmark, they provide five dimensions as tunable parameters to create the attacks, namely, 1. *Location* of the buffer being overflown 2. Using either a direct or indirect *overflow technique*. 3. Target *code pointer* to be overwritten with the 4. Address of the *attack code* to be executed, and 5. Vulnerable *function abused* to overflow the buffer. As a deception tactic, MAYAVI was invoked in *Honey Mode* to protect the 16 target *code pointer* addresses that would be overwritten with the attack code's addresses. Specifically, on an overflow attempt, MAYAVI replaced the target address with the starting address of a honey code snippet (from a honeypot of benign code) such that the program control flow never reached the attack code. Ultimately, MAYAVI was triggered 850 times, corresponding to all possible attack configurations specified by RIPE. According to the benchmark's output, each attack failed when the MAYAVI LD/ST deception engine protected the code pointers.

The ***InfoStealer*** [30] malware sample chosen for our evaluation exfiltrated necessary user credentials like browser passwords, cookies, history, and encryption keys. For this case study, the InfoStealer sample opened an encrypted browser password file, read its contents, and then transmitted them to a C&C server until all passwords were eventually stolen. We discovered 2 sensitive address ranges indicating a *read buffer* that collected the individual password bytes read from the system and a *transmission buffer* that subsequently sent the passwords to the external server. In a *Random Mode* of deception, MAYAVI aimed at deceiving memory requests that attempted to access the read buffer and redirected the requests to random address ranges via the *address randomizer unit*. As a result, even if the attacker could decrypt the extracted

Table 1: Security Analysis of MAYAVI Framework. For each malware sample evaluated, we show the number of watched (static) address ranges, the deception mode used to redirect memory requests, and the security outcomes.

Malware sample	Malware type	No. of sensitive addr. ranges (static)	Deception mode deployed	Security outcomes
WannaCrypt0r	Ransomware	5	Honey Mode	Directories protected
RIPE	Buffer Overflow	16	Honey Mode	Pointers secured
Petya	Ransomware	6	Honey Mode	Files & Directories hidden
InfoStealer	Spyware	2	Random Mode	Passwords guarded
Bad Rabbit	Ransomware	6	Honey Mode	Files protected

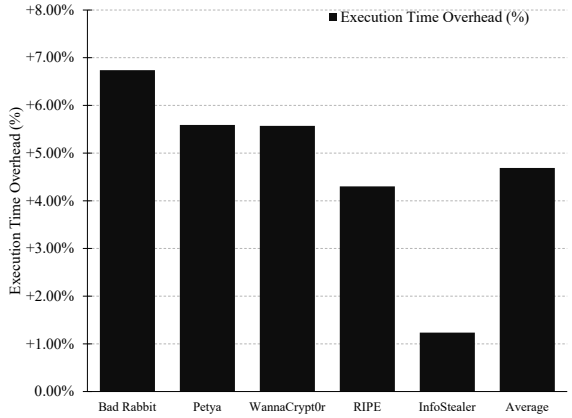

Figure 2: Execution time overheads due to MAYAVI (%) over native execution of each malware.

information, the obtained data would be useless. Consequently, MAYAVI's deception actions protected the user's passwords.

From the above case studies, it is worth noting that *the number of deception triggers is proportional to the memory requests issued by the malware accessing sensitive memory locations*. Realistically, a malware targets multitudes of vulnerable memory locations on a victim system and issues a proportionate amount of memory requests to access the sensitive data. A software-based cyber-deception strategy using techniques like API hooking [28] may impose prohibitive overheads due to heavy code instrumentation needed for deception-related operations via software. On the contrary, the proposed hardware-assisted cyber-deception framework can actively deceive all sensitive memory requests directly and transparently using hardware.

4.2 Efficiency of MAYAVI

Most cyber attackers prioritize the quick exploitation of a victim and uninterrupted execution of their malware payload without any visible discrepancies in its execution profile. Hence, for an active cyber-deception framework to be *efficient*, it is paramount to avoid detection by an adversary that analyzes its execution environment. A widely practiced approach among adversarial actors is to time their execution and spot anomalies [24]. Any deviation in the execution time from its acceptable range can potentially alert the attacker and, in the best case, discourage them from executing the malware and, in the worst case, lead to an enhanced payload capable of subverting the defenses.

To evaluate the efficiency of MAYAVI, we executed each of our malware samples on Gem5 and measured their execution times as our baseline. For the range filter, when *LBA* and *UBA* address comparisons are performed in the rising and falling edge of a clock cycle, the average latency is $0.5ns$ [31]. Upon a range filter hit, the deception mode lookup latency is $0.5ns$ for a 16-entry on-chip range filter used in our design. The *honey address generation* and *address randomizer* units add latencies of $34ns$ and $30ns$, respectively, to a memory request while accessing sensitive data locations. These latencies were modeled based on the average latencies of floating point unit operations in a modern processor (add - $2ns$, multiply - $4ns$, divide - $18ns$) [10]. The $10ns$ latency for RAND value computation is also added as specified by Intel DRNG [9]. In summary, the average latency of a *memory request being deceived in honey mode* is $\approx 35.5ns$, and *in random mode* is $\approx 31.5ns$ which are significantly lower compared to some API hooking techniques ($\approx 16.08\mu s$) [18].

Figure 2 shows the execution runtime overheads of the MAYAVI deception engine over the native execution time for each malware. The average runtime overhead across the five malware samples due to our deception engine is $\approx$ 4.69%. Our experiments show that the number of watched memory ranges and the total number of memory requests increased the overall execution time of each malware. Conclusively, the *low-performance overheads* with hardware support enables efficient and effective deception against each malware.

5 RELATED WORK

Prior research studies have explored active cyber-deception techniques to thwart memory corruption and information-stealing attacks. Honey patches [3] were proposed to tackle attackers by redirection to un-patched decoys of the known vulnerable software, which notes ineffectiveness against unknown attack vectors and backdoor threats. PhantomFS [7] proposed a decoy-file and kernel-based interface, which is futile if the attacker compromises the kernel altogether. API hooking-based software level deception [28] against ransomware and InfoStealers have reported relatively high execution time overheads to achieve deception (17 seconds [28]). These overheads are significant for a production system and could alert evasive adversaries. In contrast to these approaches, MAYAVI incurs negligible overheads, as shown in our evaluation, and provides a targeted deception on sensitive memory requests allowing it to remain concealed from the attackers.

We note that system security may often be fortified using multiple defense layers. In addition to cyber-deception, other defender mechanisms may also be utilized. Landsborough et al. [19] proposed combining autonomic computing, game theory, and cognitive and behavioral psychology to deploy decoys effectively. Foreseer [16] proposed an LSTM-based framework that reduces the detection time of the front-end malware detectors by up to 40% to support a proactive defensive framework. Prior research in hardware-based defense has studied asset obfuscation strategies like Moving Target Defense in continuous churn periods [15]. However, such mechanisms would need to be used judiciously, as they may incur high overheads due to repeated program runtime alteration (code pointers, address space, instruction layout). With frequent churns, it can become infeasible in a production environment. Other obfuscation techniques propose memory request/response rate modulation [33] and prefetching cache blocks to obliterate access patterns [12, 13]. While these are effective against memory snooping attacks, these may be detected by a sophisticated adversary with auditing capability on the memory traffic/bandwidth due to its continuous injunction of fake memory request traffic to alter memory access patterns.

6 CONCLUSION

We presented MAYAVI, a hardware-based cyber-deception design integrated with a processor pipeline's Load/Store unit. Our deception engine replaces the target addresses of suspect memory requests with honey addresses. We evaluated the effectiveness of our design using known malware samples and demonstrated how to counter them using our deception-based defense. We also show that MAYAVI can efficiently defend against malware with minimal overheads, allowing the deception framework to remain transparent to the attacker. We note that MAYAVI has paved the way towards the design of more hardware deception primitives, which may include hardware performance analyzers to falsify architectural information to an adversary, dynamic instruction editing capabilities to mislead adversaries, and instrumenting page translation mechanisms (page table, TLBs) to alter malicious memory requests transparently.

ACKNOWLEDGMENTS

This research is based on work supported by the US Office of Naval Research under grant N00014-21-1-2520.

REFERENCES

[1] Jagmeet Singh Aidan, Harsh Kumar Verma, and Lalit Kumar Awasthi. 2017. Comprehensive survey on petya ransomware attack. In *IEEE ICNGCIS*. 122–125.
[2] Ehab Al-Shaer, Jinpeng Wei, W Kevin, and Cliff Wang. 2019. Autonomous cyber deception. *Springer* (2019).
[3] Frederico Araujo, Kevin W Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. 2014. From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *ACM CCS*. 942–953.
[4] Leyla Bilge and Tudor Dumitraş. 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In *ACM CCS*. 833–844.
[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.
[6] Jie Chen and Guru Venkataramani. 2014. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *IEEE/ACM MICRO*. 216–228.
[7] Jione Choi, Hwiwon Lee, Younggi Park, Huy Kang Kim, Junghee Lee, Youngjae Kim, Gyuho Lee, Shin-Woo Shim, and Taekyu Kim. 2020. PhantomFS-v2: Dare You to Avoid This Trap. *IEEE Access* 8 (2020), 198285–198300.
[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).
[9] George Cox, Charles Dike, and DJ Johnston. 2011. Intel's digital random number generator (DRNG). In *2011 HCS*. 1–13.
[10] Srinivas Devadas. 1998. *Computer Arithmetic*. https://people.csail.mit.edu/devadas/6.004/Lectures/lect17/index.htm
[11] Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. 2007. Dynamic spyware analysis. *USENIX Annual Technical Conference* (2007).
[12] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, and Guru Venkataramani. 2019. Prodact: Prefetch-obfuscator to defend against cache timing channels. *International Journal of Parallel Programming* 47 (2019), 571–594.
[13] Hongyu Fang, Miloš Doroslovački, and Guru Venkataramani. 2022. SC-K9: A Self-synchronizing Framework to Counter Micro-architectural Side Channels. In *IEEE ASP-DAC*. 11–18.
[14] Christina Fowler, Mike Goffin, Bill Hill, Richard Lamourine, and Andrew Sovern. 2020. *An introduction to Mitre Shield - Mitre Corporation*. https://shield.mitre.org/resources/downloads/Introduction_to_MITRE_Shield.pdf
[15] Mark Gallagher, Lauren Biernacki, Shibo Chen, Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Austin Harris, Zhixing Xu, Baris Kasikci, Valeria Bertacco, et al. 2019. Morpheus: A vulnerability-tolerant secure architecture based on ensembles of moving target defenses with churn. In *ACM ASPLOS*. 469–484.
[16] Kailash Gogineni, Preet Derasari, and Guru Venkataramani. 2022. Foreseer: Efficiently Forecasting Malware Event Series with Long Short-Term Memory. In *IEEE SEED*. 97–108.
[17] N.A. Hassan. 2019. Ransomware Families. In *Ransomware Revealed*. Springer, 47–68.
[18] Shun-Wen Hsiao, Yeali S Sun, and Meng Chang Chen. 2020. Hardware-assisted MMU redirection for in-guest monitoring and API profiling. *IEEE Transactions on Information Forensics and Security* 15 (2020), 2402–2416.
[19] Jason Landsborough, Luke Carpenter, Braulio Coronado, Sunny Fugate, Kimberly Ferguson-Walter, and Dirk Van Bruggen. 2021. Towards Self-Adaptive Cyber Deception for Defense.. In *HICSS*. 1–10.
[20] Fangfei Liu and Ruby B Lee. 2014. Random fill cache architecture. In *IEEE/ACM MICRO*. 203–215.
[21] Marc Löw. 2018. Overview of meltdown and spectre patches and their impacts. *Advanced Microkernel Operating Systems* (2018), 53.
[22] Orkhan Mamedov, Fedor Sinitsyn, and Anton Ivanov. 2017. Bad rabbit ransomware. *Retrieved May* 1 (2017), 2021.
[23] Matt Miller. 2019. *Microsoft: Blue Hat conference*. https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/
[24] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. 2017. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *IEEE SP*. 1009–1024.
[25] Savita Mohurle and Manisha Patil. 2017. A brief study of wannacry threat: Ransomware attack 2017. *IJARCS* 8, 5 (2017), 1938–1940.
[26] Mahmoud T Qassrawi and Zhang Hongli. 2010. Deception methodology in virtual honeypots. In *IEEE NSWCTC*, Vol. 2. 462–467.
[27] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.
[28] Md Sajidul Islam Sajid, Jinpeng Wei, Basel Abdeen, Ehab Al-Shaer, Md Mazharul Islam, Walter Diong, and Latifur Khan. 2021. Soda: A system for cyber deception orchestration and automation. In *ACSAC*. 675–689.
[29] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2016. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE SP*.
[30] Pedro Tavares. 2021. *Redline stealer malware: Full analysis*. https://resources.infosecinstitute.com/topic/redline-stealer-malware-full-analysis/
[31] Mohit Tiwari, Banit Agrawal, Shashidhar Mysore, Jonathan Valamehr, and Timothy Sherwood. 2008. A small cache of large ranges: Hardware methods for efficiently searching, storing, and updating big dataflow tags. In *IEEE/ACM MICRO*. 94–105.
[32] John Wilander, Nick Nikiforakis, Yves Younan, Mariam Kamkar, and Wouter Joosen. 2011. RIPE: Runtime intrusion prevention evaluator. In *ACSAC*. 41–50.
[33] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. 2017. Camouflage: Memory traffic shaping to mitigate timing attacks. In *IEEE HPCA*. 337–348.