Download More RAM: Dismantling Windows Operating System Defences with Mischievous Memory

Sam Collins
University of Birmingham

Tom Chothia
University of Birmingham

William Burgess
University of Birmingham

Marius Muench
University of Birmingham

David Oswald
Durham University

Abstract

Virtualisation-Based Security (VBS) is the cornerstone of modern Windows desktop defences, relied upon by both the operating system and third-party software, with a virtualised secure kernel providing strong security guarantees against even privileged attackers. In this paper, we introduce *Download More RAM*, a software-only memory aliasing attack that breaks these guarantees without physical access. On systems running the most common consumer DIMMs, our attack allows arbitrary memory read/write, letting a privileged user compromise the OS at every level, including the secure kernel, Hypervisor Enforced Code Integrity (HVCI), and all defences it provides. With this access we develop a series of case study attacks targeting VBS-protected processes, Windows Defender, anti-virus & EDR software, and game anti-cheats. Our work breaks the strongest security guarantees offered by the Windows OS, questioning key trust assumptions on such systems. Microsoft have assigned CVE-2026-23670 to our findings and issued a patch that partially mitigates our attack.

1 Introduction

Windows operating system security has seen significant advancements, with the addition of a wide range of hardware-supported security features, providing protections even against privileged attackers. Windows 11 explicitly aims to protect against a malicious process with admin privileges. Kernel code must be signed by Microsoft [38] denying access to even high privilege attackers. If an attacker does compromise the kernel regardless, a further level of defence is provided by *Virtualization-Based Security (VBS)*, and its suite of defences such as *Hypervisor Enforced Code Integrity (HVCI)* [37, 39]. VBS uses Hyper-V to run the Windows Kernel in a lightweight VM, which is monitored by a *Secure Kernel* running in a different isolated container. Therefore, an attack that breaches the regular kernel cannot access the core of Windows security and would be detected and stopped. A range of other technologies protect Windows OS against an attacker (or careless user) with admin access. Process Protection Light (PPL) [41] allows for processes that cannot be terminated by an admin user, often used for Anti-Virus and EDR. Mobile Device Management makes it possible to remotely manage a machine, denying an admin user the ability to disable anti-virus or install applications of their choosing.

The *Download More RAM* attack presented in this paper defeats all of these defences, giving a privileged user control over the full system including the core OS as well as the secure kernel. We show how a Windows attacker, with admin privileges, can break out of the standard Windows container into the secure kernel, arbitrarily patch security critical code, and disable defences, allowing blocked kernel drivers from previous malware campaigns to run again.

Underlying our attack is the fact that all processes, no matter their protection, use the same physical memory. We exploit this using a memory aliasing technique building upon the recent BadRAM attack [13]. BadRAM showed that physically removing memory from a machine and reprogramming it to report double its size opens a new avenue for attacks. On such reconfigured memory, the top half of the addresses become aliases for the lower half, and an attacker can bypass the read and write protections on the lower half of memory by accessing the aliased address via the corresponding upper half address. This unrestricted access has been used to attack Trusted Execution Environments (TEEs) on Linux, assuming an attacker with physical access and kernel-level privileges [12, 13, 34].

In this paper we show, for the first time, that memory aliasing attacks can be carried out using just software, with no physical access to a machine, for a significant share of aftermarket consumer RAM[1]. We also show how Windows can be configured to run with aliased memory and how an attacker, without arbitrary kernel access, can then access that aliased memory. We do this by truncating the memory available to Windows to the lower half of memory and then allocating the alias addresses of the upper half as a RAM disk.

[1]We note that past work [13, 34] speculated that software-only attacks might be possible, but did not demonstrate them, end-to-end, in practice.

Accessing the aliased memory as a RAM disk is brittle: we cannot write to individual addresses and instead can only write larger sections, which includes file metadata. In initial testing, this often led to fatal system crashes and the infamous blue screen of death. Therefore, we use our limited access to disable driver signature enforcement checks in the Windows Kernel and in the Secure Kernel. We then load previously blocked vulnerable drivers[2] to give us arbitrary read and write to any system memory providing a powerful and stable attack primitive.

To assess the prevalence of potentially vulnerable machines and the impact of the *Download More RAM* attack, we survey a representative set of consumer DIMMs and provide four case studies defeating both Windows and third-party defences. We find three DIMM manufacturers vulnerable to the attack, with an estimated market share of over 55% and 70% on the high-performance and gaming segment, respectively.

In summary, the contributions of this paper are:

- We demonstrate the first software-only memory aliasing attack from a privileged user, eliminating the need to assume kernel-level access and physically removing and re-installing memory on the target.
- Finding Windows systems to be unbootable with aliased memory, we find novel ways to stabilise victim Windows systems, access aliased memory, and decode scrambled memory contents.
- We reverse engineer the Windows code integrity ecosystem and use our attack to disable the vulnerable driver blocklist – leading to clean arbitrary access to the entirety of physically installed memory.
- Building on our attack primitives, we provide a range of case studies: patching VBS protected processes, disabling Windows defender, third-party EDR and patching game anti-cheats. We also show how the attack can be executed as a one-click script, with no user interaction.

Coordinated Disclosure We have disclosed our findings to affected parties. Microsoft confirmed our attacks, have issued CVE-2026-23670, and pushed mitigation included in the April 2026 Security update. We have disclosed the weakness to all affected RAM manufacturers. Corsair confirmed the attack and have supported our further investigations, while discussions with other RAM manufacturers are ongoing. EPIC Games and RIOT games acknowledged the impact of our findings on their anti-cheat systems and awarded bug bounties. We have also disclosed to Sophos, who have noted the submission as valuable and are discussing further steps internally.

[2]We note that we do not need to use blocked vulnerable drivers as our attack primitive would allow us to load unsigned drivers that we had written ourselves. We use known vulnerable drivers with well documented primitives, often used in past malware campaigns, as they make the overall attack development easier.

2 Background

2.1 Windows OS Security

Windows has historically been one of the most targeted operating systems by malware [2]. Consequently, Microsoft enforces strict kernel-level code-signing requirements known as Driver Signature Enforcement (DSE). All kernel-mode drivers must be digitally signed by Microsoft, and unsigned drivers can only be loaded when test-signing mode is enabled—typically for development.

In tandem with its code-signing requirements, Windows runs the Kernel Patch Protection system, commonly known as PatchGuard, to prevent modification of critical kernel structures and code. Because PatchGuard operates within the kernel, where malicious actors are assumed to have comparable privileges, it conceals its execution by running within legitimate system threads and piggybacking inconspicuous system functions as its entry points. Both DSE and PatchGuard suffer a significant flaw, they protect the kernel from the kernel, with no guarantee of a strict boundary between them and a potential attacker. Since the release of Windows 10 in 2016, the OS has also supported Virtualisation Based Security, introducing strict hypervisor enforced boundaries to prevent the kernel from high privilege and even physical attacks. On supported hardware, VBS is enabled by default on distributions of Windows 11.

Windows Hyper-V Hyper-V is Microsoft's native Type 1 hypervisor, responsible for creating and managing virtual machines (VMs) on Windows systems. As a Type 1—or "bare-metal"—hypervisor, it operates directly on the system's hardware, below the main OS. This architecture minimizes performance overhead and enforces strong hardware-level isolation between individual virtualized environments.

Although Hyper-V was largely developed for server and cloud virtualization, it now serves as the fundamental enabling technology for Windows VBS. On modern Windows systems, the primary 'user-facing' operating system itself runs within a Hyper-V partition. Critical security services are subsequently hosted in a separate, isolated partition, known as Virtual Trust Level 1 (VTL1), see Figure 1. This partition is further subdivided into the Secure Kernel and the Isolated User Mode. This configuration establishes a hardware-enforced security boundary that the Windows kernel alone cannot guarantee.

Windows Virtualization-based Security. Windows Virtualisation Based Security (VBS) is a security system underpinned by Hyper-V. VBS uses Hyper-V to create isolated regions of memory for security-sensitive processes, protecting credentials, kernel integrity, and other critical OS components. Several features comprise the VBS ecosystem and can be enabled/disabled in a plug-and-play manner through the OS' configuration.

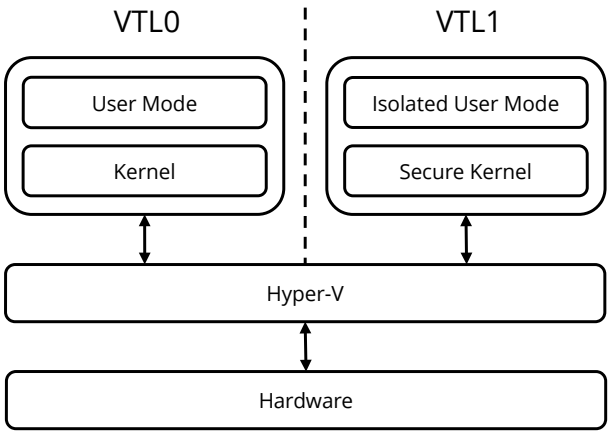

Figure 1: Windows Virtual Trust Levels Visualised.

Hypervisor Enforced Code Integrity HVCI was introduced to address persistent challenges in Windows kernel security, notably "Bring Your Own Vulnerable Driver" (BYOVD) attacks. While Microsoft mandates code signing for kernel code, attackers can exploit legitimate but vulnerable signed drivers to load arbitrary malicious code into the kernel. This technique is commonly leveraged by rootkits, ransomware, game cheats, and other advanced persistent threats to gain elevated privileges, disable protections such as Anti-Virus or EDR, and evade detection. To counter BYOVD attacks, Windows maintains a block list of known vulnerable drivers and will check all loaded drivers against this list.

HVCI attempts to prevent this by relocating code integrity enforcement from the standard Windows kernel to the isolated, hypervisor-protected secure kernel. This hardware-isolated environment performs strict validation of all code pages, including verifying digital signatures and enforcing memory protection policies. HVCI strongly enforces nx protections (W⊕X) by removing page table authority from the normal kernel. This prevents the introduction of arbitrary, executable code from an attacker in the kernel. HVCI further mandates checks on the known reputation of loaded driver images; as a requirement to run HVCI, Microsoft's vulnerable driver blocklist must be enabled to prevent the loading of known buggy/vulnerable drivers.

2.2 RAM Configuration and Memory Aliasing Attacks

A DIMM's configuration is stored on an Erasable Programmable Read-Only Memory (EEPROM) chip using the Serial Presence Detect (SPD) standard. This includes the type of RAM (DDR3, DDR4, DDR5, etc.), frequencies, voltage, optional overclocking profiles, and capacity among other information [24, 25]. The capacity of a single RAM stick is organised hierarchically, top to bottom: DIMM, Rank(s), DRAM Chips, Bank Groups, Banks, Rows, and Columns.

The fact that the memory controller implicitly trusts the correctness of SPD data was first exploited by De Meulemeester et al. in the context of x86 [13] and subsequently extended to RISC-V systems [34]. However, both papers assume one-time physical access to extract the DIMMs from the machine, remove write protections, alter SPD contents and then replace the DIMMs. Though De Meulemeester et al. noted that they encountered an unlocked UDIMM module, they did not demonstrate how to practically exploit this observation. For this reason, their attacks remain limited in scope to trusted execution environments (such as AMD SEV-SNP and RISC-V PMP/Keystone), where physical access, and kernel-level compromise falls into the adversary model. Along similar lines, the Battering RAM attack showed that memory behaviour can also be altered at runtime with a custom interposer [12], again requiring physical access to the target system to insert the interposer between the DIMMs and the motherboard.

3 Attacker Model

Following a man-at-the-end attack scenario [1], we assume an attacker with local admin privilege on a Windows 11 system, fully secured with all advanced protection features in-place (i.e., Secure Boot, DSE, PatchGuard, VBS/HVCI). We further assume that the target system leverages DIMMs without SPD write protection enabled; As we will show, such system configurations are common on the consumer market.

The goal of the attacker is system-level compromise, bypassing VBS and achieving arbitrary memory disclosure and modification at all levels, including the Windows kernel, secure kernel, and the Hyper-V hypervisor, even in cases where memory is marked inaccessible or immutable.

In contrast to prior work [12, 13, 34], we do not assume physical access, capabilities to re-flash system-level firmware, or the presence of any software vulnerabilities in the Windows kernel, installed drivers, or third-party components. As such, the attacker model applies, for instance, to rogue system administrators, as well as remote attackers and malware with elevated privileges. We note that our attacker model is significantly weaker than the threat model introduced by VBS [39] or past memory aliasing attacks [12, 13, 34], all of which further assume that the kernel is already compromised.

4 Attack Overview

In this section we provide an overview of *Download More RAM*, our attack leveraging software induced memory aliasing to break the isolation provided by virtualization-based security on Windows 11 endpoint machines. *Download More RAM* creates an arbitrary read and write primitive for physical memory, bypassing code integrity checks, as well as all MMU-based memory protections.

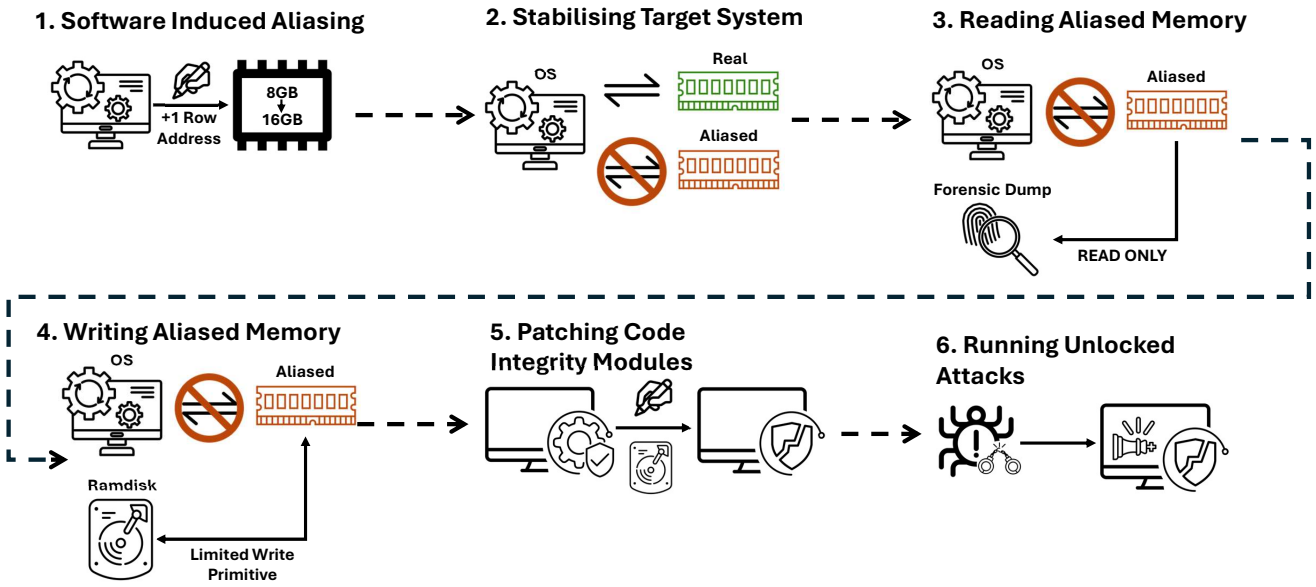

Figure 2: Overview of *Download More RAM*. An attacker with local admin access *(1)* introduces memory aliasing from software, *(2)* stabilises the system to prevent crashes due to the induced aliasing, *(3)* uses commonly available forensics tool to read-out aliased memory, *(4)* utilises ramdisk utilities to create a limited write primitives, allowing to *(5)* disable critical code integrity checks, leading to *(6)* extended attack capabilities completely bypassing virtualization-based security.

The full attack chain is shown in Figure 2 and consists of six phases: (1) software induced aliasing, (2) stabilising target system, (3) reading aliased memory, (4) creating an aliased memory write primitive, (5) patching code integrity modules, and (6) running unlocked attacks.

1) Software Induced Aliasing. *Download More RAM* begins with setting up the currently installed DIMM(s) on the target system. Our attack reconfigures a DIMM fully from software by rewriting is SPD EEPROM contents via the System Management Bus (SMBUS) accessible through the I^2C subsystem. On DIMMs with disabled SPD write protection, we increment the configured number of row addressing bits, effectively causing the DIMM to report twice the amount of physically present memory. Accessing this memory will cause the highest order row addressing bit to be disregarded in any access – effectively creating an alias between the lower and higher half of memory on that DIMM.

2) Stabilising Target System. The presence of aliased memory on a system causes significant reliability issues and we could not find an aliased memory configuration in which a system was able to progress through the early boot stages. Hence, we leverage specialised boot parameters (e.g., `removememory`) to limit the memory used by the victim. By disallowing the use of the upper memory half, we ensure that the OS can operate stably without any exceptions or crashes induced by the aliasing.

3) Reading Aliased Memory. While the victim machine is now stable, we have explicitly instructed Windows to not use the aliased memory. Therefore, we cannot trivially access this memory. However, we identified a set of kernel functions operating directly on the physical memory, effectively bypassing the restrictions introduced by step (2). Further analysis revealed legitimately signed drivers using one of these functions with user-mode controlled arguments, allowing us to read-out the the aliased memory for further analysis.

4) Enabling Aliased Memory Writes. While the drivers identified in step (3) allow reading of aliased memory, they do not provide a direct write primitive. To modify memory contents, we instead use a ramdisk utility, which makes it possible to allocate a virtual filesystem over the aliased space the OS considers unused. We can now directly write to memory through direct file accesses. Unfortunately, due to metadata, filesystem layout, and dynamic memory changes, this approach is likely to overwrite critical data structures as side effect, risking a system crash. Hence, we limit the size of the allocated filesystem and only overlay target victim code to expand our write primitive in the following steps.

5) Patching Code Integrity Module. Using the limited write primitive of the last step, we patch the module ensuring code integrity and implementing driver signature enforcement`skci.dll` (for secure kernel code integrity). Our patches disable the driver revocation checks, rendering the driver blocklist maintained by the secure kernel ineffective.

6) Running Unlocked Attacks. With the modifications from step (5), we can now load previously blocked vulnerable drivers. As a result, priorly mitigated attacks can now be launched again against the victim. Additionally, utilizing vulnerable drivers providing physical memory read/write primitives, *Download More RAM* can now directly access and modify aliased memory without any restrictions, *bypassing all memory protections and completely defeating virtualization-based security*.

5 Memory Aliasing without Physical Access

We first show how an attacker without one-time physical access and kernel-level privileges can induce memory aliasing to a victim system and then analyse the effects of different aliasing approaches.

5.1 Rewriting SPD Contents from Software

To demonstrate the threat posed by our attack against Windows endpoints, we first remove a significant limitation of prior work—the need for physical access. Our attack targets DIMMs without SPD write protection. As we will show, these are widely deployed on the consumer market (i.e., on systems which are likely to run Windows).

On Linux, writing to an unlocked SPD chip is trivial via the suite of i2c kernel modules complemented with packages such as `i2ctools` or `i3ctools` [34]. However, Windows does not expose similar functionality by default and, as outlined in Section 2.1, running kernel code on Windows requires a code signing certificate from Microsoft. Without this high-privilege/low-level access we are not able to interface with the bus connecting to the SPD chips. We identify two tractable solutions to this issue on Windows: (a) using a Bring-Your-Own-Vulnerable-Driver (BYOVD) attack leveraging a weakness in signed kernel drivers to obtain low-level access, and (b) finding a signed application that lets us directly interface with SPD chips.

Access via Known Vulnerable Drivers. With the recent rise of BYOVD attacks, a range of tools implement capabilities to gain kernel-level access by abusing legitimately singed drivers containing known vulnerabilities. Although this technique is common in malware [42], it also found its application in more benign communities, such as overclocking and modding, to support custom tooling. We identified two existing tools that use known and blocked vulnerable drivers to write to DDR4 SPD chips: RWEverything [27] and DDR4 XMP Editor [21]. Although these tools serve no innate malicious purpose, the drivers they rely on are blocked on Windows 11 systems by the Vulnerable Driver Blocklist due to their use in past malware campaigns. Users wishing to run these tools must therefore first disable a key security feature first. This limitation does not affect applications such as game hacking, but it does prevent other attack scenarios including the one presented in this paper.

Access via Legitimate Applications. Due to the niche requirements of our attack, we found only a single application for reading/writing SPD chips on Windows accompanied with a driver not included in the vulnerable driver blocklist. Thaiphoon Burner [52] is a tool designed for overclocking/tweaking RAM profiles manually from the OS[3] Although the most recent version of Thaiphoon Burner allows operation under fully updated Windows machines, the drivers in previous versions are blocked. Our reverse engineering efforts suggest that the tool uses a recent version of the msio.sys driver, developed by MSI for use in *afterburner* - MSI's RGB lighting and hardware control utility.

Hybrid Approach Upon the discovery that Thaiphoon Burner uses a legitimate third party driver developed by MSI, we decided to develop our own SPD read/write tool. To build this tool we used (a) the decompilation of the msio.sys driver, reversed with IDA (b) DeviceIoControl[4] traces captured from Thaiphoon Burner communicating with the driver (c) the relevant Joint Electron Device Engineering Council (JEDEC) specification document [26].

Regardless of the technique used, an attacker needs to restart the victim machine after modifying the SPD contents, as the SPD data is read during boot to configure the system. This restart can be done from software in Windows with additional attack code set to automatically run after the restart.

We note that restarting alone is not sufficient on all systems. Some UEFI/BIOS implementations cache the SPD data and not read them on each boot. To overcome this, we additionally alter the vendor information segment of the SPD, including the serial number. Due to these changes, the BIOS believes that a new DIMM has been installed and forces a full re-read of the SPD.

5.2 Effect of Aliasing

When rewriting SPD contents, an attacker can choose between two candidate values for modifying the reported size to create aliases—the row address bits and the column address bits. Because of DIMM geometry, the system uses higher column addresses before higher row addresses. As a result, if we increment the column-address bits, the system's physical addresses will alternate between real and aliased memory on a per-row basis: the lower half of each row will map to real memory, and the upper half will map to the aliased region.

[3] Similar tools exist as UEFI applications, but we consider them out of scope as they require to be signed under SecureBoot.

[4] Analogous to ioctl on Unix systems.

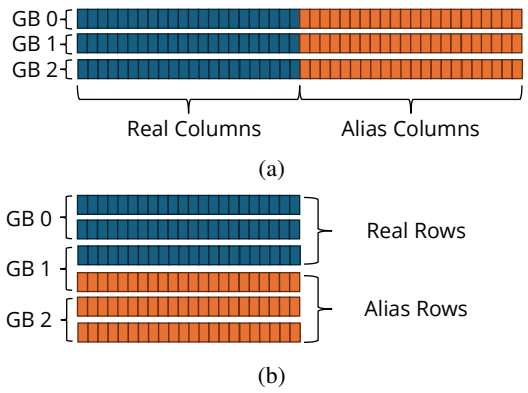

Figure 3: The geometrical difference in incrementing *(a)* the column address bits and *(b)* the row address bits.

This would result in a system configuration with no large contiguous region of regular memory or of aliased memory, with the two spaces equally interleaved, as shown in Figure 3a.

Alternatively, an attacker can increment the row addressing bits. Since the topmost DIMM addresses correspond to topmost row addresses, this will cause alternation between real and aliased memory on a per bank basis. For a victim system, this creates two contiguous regions with the lower half being real and the upper half being aliased memory (cf. Figure 3b).

The second option, incrementing the row-address bit, creates a much cleaner separation between the two spaces. We therefore adopt this as our aliasing primitive, following established techniques [13, 34].

6 The Download More RAM Attack

With aliased memory in place, we now describe the further attack steps of *Download More RAM*. Although our work assumes the introduction of memory aliases purely from software, it is noteworthy that all techniques described in this Section also apply directly for memory aliased induced via other means (e.g., via physical access as in [13]).

6.1 Stabilising Windows OS

Introducing memory aliases alone without further mitigation would render a system inoperable, as the underlying operating system would frequently access aliased memory, resulting in unintended overwrites and a crash.

We empirically verified this by attempting to boot windows on DIMMs in various aliasing configurations[5], as well as multi-DIMM configurations with one aliased DIMM installed alongside one or several regular DIMM(s). In all cases, the system was not able to boot up due to various unhandled exceptions triggering the "Blue Screen of Death" [6].

```
bcdedit /set truncatememory 0x200000000
bcdedit /set removememory 8192
```

Listing 1: Example commands to restrict memory usage from 16GB to 8GB on Windows.

Following BadRAMs intuition of adjusting the operating system's kernel parameters [13], we investigated windows boot configuration options. We found two parameters available to Windows' Boot Configuration Data Editor in user space (`bcdedit`): `truncatememory` and `removememory`. The former takes an address as its input and prevents the OS from allocating memory at any address higher than this value, while the latter takes a value in megabytes as input and removes that amount from the back of available physical memory. We show the example of configuring a 16GB aliased system to only use the lower 8GB using these parameters in Listing 1.

Although both parameters allow for a fully booting and stable Windows system under aliased memory, we note that only `removememory` is secure boot compliant and can be used on a VBS-enabled system as presented in our attacker model.

DIMM-based DoS Attacks On systems with rewritable SPD chips a system could be completely disabled by rewriting the EEPROM. With admin access on the machine, an attacker could, for instance, increment the number of DIMM columns; simple memory test such as reading and writing to block of or memory would still work but once the machine started to load the kernel, the memory would become corrupted and Windows would blue screen. While this could be fixed by replacing the memory, it would be a difficult attack to diagnose, as all hardware would pass basic BIOS tests.

6.2 Reading Aliased Memory

As an unfortunate side-effect of restricting the memory used by Windows, this memory is now also inaccessible, including to an admin-level user of the system. In other words, an attacker with admin privileges, as presented in our threat model, will not be able to access the aliased memory without further exploitation. This is in stark contrast to prior BadRAM attacks [12,13,34], which assume a root user on Linux accessing physical memory via attacker-controlled kernel modules.

On Windows, functions operating on the physical address space are reserved for kernel operation only. We identified three kernel functions allowing direct allocation of physical memory, `MmMapIoSpace`, `MmAllocateContiguousMemory`, and `MmAllocatePagesForMdl`. Of these, `MmMapIoSpace` is the only one that accepts both a starting physical address

[5] 4GB→8GB, 8GB→16GB, 16GB→32GB, and 32GB→64GB.

[6] The BSOD is equivalent to a unix kernel panic.

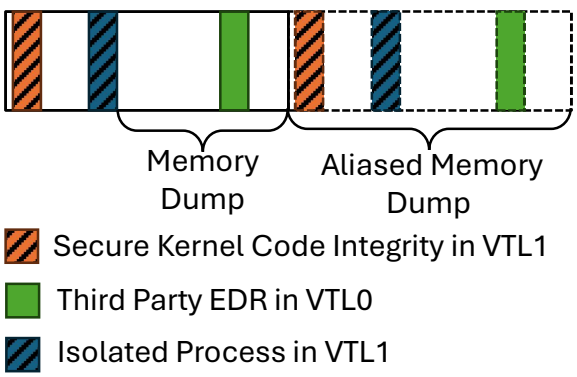

Figure 4: The memory layout of an aliased system showing the scope of access given by our custom winpmem utility.

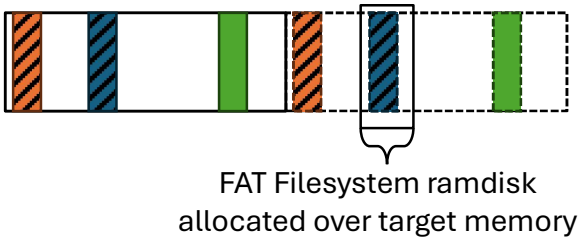

Figure 5: The memory layout of an aliased system showing how a ramdisk is allocated only over a target slice of memory.

and a size, making it the most suitable for *Download More RAM* attacks. The other two functions allow the caller to specify a size, but not an exact physical location; additionally, `MmAllocatePagesForMdl` zero-fills allocated memory which is likely to disrupt the system when used over aliased memory ranges.

While we confirmed that `MmMapIoSpace` is able to hand-out aliased memory to a kernel-level attacker (see Appendix A for details), a practical attack requires a legitimate application and kernel driver leveraging this function with user-controlled inputs. We identified *WinPmem* [8] as one of such applications, an open-source forensic tool for creating live memory dumps of an entire running system. Most importantly, its drivers are signed, but the usermode program can be reprogrammed freely to suit the specific needs of the user. Inspecting its source code further revealed that boundary checks on physical memory ranges are solely implemented in the user-mode application, alongside with limitations on the size of memory dumps. We therefore created our own custom version of winpmem which removed these safety checks and, thus, allowing us to dump any region of aliased memory, enabling the next steps of the *Download More RAM* attack. While our proof-of-concept uses this particular application and driver, reading aliased memory could be achieved with any similar tool and driver failing to enforce allocation boundaries in kernel-space.

Interpreting aliased memory dumps. When comparing dumps of unaliased memory with dumps of aliased memory, we found that their contents are not equal. Two mechanisms require us to further process the aliased memory dump: DDR data scrambling and virtual-to-physical address translation.

Since DDR3, a lightweight component called the scrambler applies a pseudo-random XOR pattern to data before it is sent over the memory bus. The scrambler key is address dependent, which leads to a mismatch between used scrambling key and actual address when reading from aliased memory. More detail, when reading address x from aliased memory, its actually contents have previously been stored with the corresponding scramble key k^x. However, the hardware will instead decode the retrieved data with with key k^y, since the chipset believes that it is reading address y. We note that this memory scrambling is not meant to provide cryptographic confidentiality, but is designed as an electric signal reliability measure, as it reduces repeating bit patterns which would create stronger electro-magnetic interference peaks leading to signal integrity issues when transferring data on the memory bus [4].

To decode scrambled memory, we scan the memory for large blocks of one repeating 32-bit pattern, as this will likely correspond to zero-initialised memory. By applying such identified patterns to the different regions of the aliased memory dump, we can restore the original, physical memory contents. To validate the correctness of the memory decoded in this way, we inspect its contents for Executable (PE) headers, human readable data, and known Windows fingerprints.

However, even after full unscrambling of an aliased memory dump, it does still not match the memory as seen when reading out the unaliased memory available to the memory dumping utility. This is due to the Windows kernel running in VTL0, rather than baremetal. As such, the "physical" memory as seen by the dumping utility when accessing unaliased memory is actual virtual memory undergoing further address translation. This also prevents unaliased memory dumps from exposing VTL-1 or Hyper-V contents, further solidifying the VBS security guarantees. However, for aliased memory, the dump provides a view on the actual raw physical addresses without any virtualisation and as seen by the hypervisor. As such, it contains all memory contents, potentially also including sensitive secrets guarded by the Secure Kernel or Hyper-V. We visualise these differences in Figure 4.

6.3 Writing Aliased Memory

Once the Windows operating system marks a memory region as unused, via available boot parameters, it is intentionally difficult to access. Furthermore, legitimate applications that provide configurable kernel-level write access are far less common than forensic read tools.

We identified ramdisk utilities as potential avenue to enable *Download More RAM* attacks. Commonly used for IO intensive tasks, ramdisk applications allow a user to partition a chunk of memory and use it as a volatile disk. Under normal operation, on a system behaving as it should, this poses no security threat – the ramdisk should only be able to allocate a drive in unused space. However, by running a system with aliased memory, we have broken the assumption that a physical address can only exist once.

For our attack's implementation, we leverage Primo Ramdisk [46], one of many ramdisk utilities commercially available. Besides the expected functionality, Primo Ramdisk also provides a feature for so-called "*invisible memory*". In reality, this "invisible memory" is any section of physically available memory inaccessible by the OS, such as any address over 4GB on a 32-bit system. This also encompasses physical memory partitioned off with a boot parameter as introduced by *Download More RAM*. Since the operating system, and also any application running on it, believes that the aliased half of memory is unused and empty, we are able to allocate ramdisks over the aliased half of memory.

Unfortunately, Primo Ramdisk has no option for creating a 'raw' partition to access, instead requiring the initialization of a filesystem in the allocated space. While this does not require formatting the memory, allocating filesystem headers over in-use, aliased memory will cause notable side-effects, potentially a full system crash or freeze. Large allocations with complex filesystems, such as NTFS, cause significantly more side-effects than small allocations using a simple filesystem like FAT. For example, attempting to allocate a full 8GB disk over the entirety of the aliased region always resulted in a BSOD.

However, Primo Ramdisk does allow us to specify a trailing space in the aliased region before ramdisk allocation, as well as the size for the ramdisk. Combined with the read primitive introduced in Section 6.2, this allows us to first find target memory and then allocate a small ramdisk over it, as shown in Figure 5. Using the created ramdisk, we can now directly modify physical memory contents by writing to its files. Notably, this bypasses all memory protections and virtualisation-based security. While powerful, this approach still introduces side-effects (e.g., by writing file meta data).

We found that the FAT filesystem, being the most lightweight, had the least effect on memory around a target. Importantly, the filesystem headers are predictable and the filesystem does not use advanced features like journaling introducing additional noise. Thus, we can easily find a place in memory where it is safe to allocate the FAT ramdisk.

To leverage the ramdisk-based write primitive, we developed a patching application automating memory modifications by: *(a)* allocating a ramdisk over the target memory region, *(b)* saving the disk's contents *(c)* descrambling, patching, and rescrambling the saved image, and *d* reinitializing the ramdisk with the updated memory. While this approach may potentially corrupt the contents of dynamically allocated and updated memory (if that memory changes in the time it takes us to complete steps b to d) it enables static code modification for key integrity modules, as we will show in the next subsection.

6.4 Patching Code Integrity Modules

Download More RAM's ramdisk technique provides arbitrary but unreliable access to aliased memory; during initial testing, we were rarely able to patch victim code and data on the first attempt without causing a system crash. However, after reconnaissance and leveraging our patching application developed in Section 6.3, we are able to allocate ramdisks at suitable offsets relative to victim modules without corrupting critical structures or code, allowing the creation of patches without unintended side effects. However, modifying arbitrary memory contents remain challenging.

We therefore develop a technique to pivot from the limited write primitive to an arbitrary read/write against live memory. In detail, we *(a)* use the ramdisk to patch a single consistent target—the OS' code integrity modules, *(b)*, load an otherwise blocked vulnerable driver, and *(c)* leverage this driver to achieve clean access to aliased memory, similar to classic BYOVD attacks.

Reverse Engineering Windows Driver Blocklisting. To identify which parts of the victim module to patch, we require detailed understanding of the inner workings of Windows' driver signature enforcement and integrity checks.

When a Windows system starts, the boot manager runs winload.efi, which uses Hyper-V to start the Windows kernel (ntoskrnl.exe) in a partition (VTL0). Winload.efi also creates a second partition (VTL1) for the secure kernel that executes the code in securekernel.exe. When a new driver is loaded, ntoskrnl.exe uses the library *skci.dll*, running in the VTL1 secure kenrel, to check the driver's integrity [47]. Therefore, we must understand how these checks are implemented and how integrity is enforced; However,*skci.dll* is a windows internal library without public documentation, requiring us to reverse engineer the library. We highlight our key findings in Table 1 and expand on them in the following.

First, `CiInitialize` function is called on boot and loads the list of blocked vulnerable drivers. We found that list contains three parts: *(1)* hashes of blocked drivers, *(2)* hashes of revoked driver signing certificates, and *(3)* hashes of revolved time authenticity signing certificates. Then, when a driver is loaded during runtime, its integrity is checked the secure kernel with the function `SkciValidateImageData`, with similar functions to check headers and memory pages. This function also checks if the loaded driver is signed and allowed to be loaded by checking its hash, signing certificate and time authority certificate against the revoke list.

Function	Library	Purpose
SkciValidateImageData	skci.dll	Validate driver layout & integrity
I_MinCryptIsHashRevoked	skci.dll	Determine if hash is contained in local blocked driver list
I_MinCryptIsCertificateHashRevokedV2	skci.dll	Determine if hash is contained in local blocked driver list
I_MinCryptVerifyCertificateWithPolicy2	skci.dll	Validate certificate
SIPolicyBlockErrorCodePerPolicy2	skci.dll	Check for cloud-based revocation of driver

Table 1: Key functions implementing integrity checks for detecting revoked drivers on Windows. *Download More RAM* patches the bottom four to allow loading of previously blocked vulnerable drivers.

The check of the driver's hash is implemented by `I_MinCryptIsHashRevoked` and certificate validation is carried out in `I_MinCryptIsCertificateHashRevokedV2`. Both of these look up the hash type and then use `bsearch` to determine if the corresponding hash is contained in the revoke list. Additionally, we examined `skci.dll` for functions returning the error code associated with blocked drivers (*0xC0000428*) and found the additional functions `SIPolicyBlockErrorCodePerPolicy2` and `I_MinCrypt VerifyCertificateWithPolicy2`. The former implements Microsoft "Smart App Control", which provides additional cloud based revocation and integrity checks [36], while the latter verifies the general validity of a certificate including its expiry date.

Re-enabling blocked drivers. Based on our reverse engineering results, we decide to patch all functions validating the certificate and revocation status for a driver being loaded. This results into 5 functions requiring modification.

To determine the location of the functions in physical memory, we inspect obtained memory dumps and make a key observation: The *Secure Kernel Code Integrity (skci.dll)* library is always located in the same physical location in memory across reboots. This observation enables us to deploy direct patches to *skci.dll* after an initial memory probe of the system.

After identifying the physical placement of the library, we use the limited write primitive discussed in Section 6.3 to provide patches with minimal invasiveness. For `SIPolicyBlockErrorCodePerPolicy2`, we change the returned error code to 0, which internally indicates success and, hence, removes the Smart App Control protections. In all other cases, our patches cause the target functions to directly return the value 0 (i.e., *success*)[7].

With the patches in place, we can now load *any* previously blocked vulnerable driver onto our system, **effectively bypassing a cornerstone of HVCI**. As Microsoft frequently blocks vulnerable drivers directly after their discovery, the capability to load arbitrary vulnerable drivers is invaluable to potential attackers.

[7]We note that an alternative technique would be to patch the block list itself. However, due to its changing nature, this would require frequent updates of our attack to identify the list in memory.

6.5 Running Unlocked Attacks

Download More RAM "unlocks" hundreds of known vulnerable drivers previously prevented from loading by the blocklist, re-enabling many attacks which have been impossible since their initial detection. Unsurprisingly, the arsenal of vulnerable drivers and derived attack scenarios is significant [42], and we will explore example end-to-end attacks in Section 7.

Beyond this, we note that vulnerable drivers achieving an arbitrary memory read/write (e.g., *RTCore64.sys*/ CVE-2019-16098) have particularly severe consequences under a *Download More RAM* attack. While originally only being able to access the "physical" memory available to VTL-0, these drivers can now directly read and tamper aliased memory. As a result, an attacker can access the complete real physical memory of a victim system without any permission checks. This completely invalidates all isolation guaranties provided by VBS, enables a wide-ranging set of attacks, and provides even stronger capabilities than specialised Direct Memory Access cards leveraged by physical attackers.

7 Evaluation

Our evaluation aims to answer the question *"do* Download More RAM *attacks pose a realistic threat to Windows endpoint systems"*? For this, we first survey a range of consumer DIMM modules for disabled SPD write protection in Section 7.1. Then, in Section 7, we present four showing practical applications of *Download More RAM*.

7.1 Affected RAM Models

To understand breadth of our attack surface, we examine several DDR4 and DDR5 UDIMMs and check their write protection status. We show the full results of this survey in Table 2. While we refrain from exhaustive testing due to budget constraints[8], we evaluate some of the most popular end-user DIMMs from a range of vendors.

JEDEC states that the first two SPD data blocks should be write protected on UDIMMS, therefore, even standard-

[8]Between research design and execution, the prices for consumer DIMM have experienced a significant increase. We did not test ECC DIMMs, but note that they do not provide protections against BadRAM attacks [13].

DDR version	Manufacturer	Product line	Part no.	Size (GB)	Protection
DDR4	Corsair	Vengeance	CMK32GX4M-2A2666C16	16	○
			CMK32GX4M-1D3000C16	8	○
	G.Skill	Aegis	F43200C16-8GIS	8	○
		T.Z Royal	F43200C16-8GTRS	8	◐
	ADATA	XPG	DDR4 3200 2OZ	8	○
	HyperX	Fury	HX432C16F-B3K2	8	◐
	Crucial	-	CT8G4DFS8266.M8FE	8	◐
	Kingston	-	99U5678-029.A00G	8	◐
DDR5	Corsair	Vengeance	CMK32GX5M-2B5200C40	8	○
	Crucial	-	CT32G56C4-U5.C16D	32	◐
	G.Skill	T.Z NEO	F56000J30-40G32G	32	◐

Table 2: SPD write protection status of different UDIMMs from multiple vendors across DDR4 and DDR5. ○ indicates "all blocks unprotected" while ◐ indicates partial protection. Partial protection is enough to prevent our attack.

compliant *partial write protection* of the SPD would prevent *Download More RAM*, as these blocks store the core DIMM configuration values. However, we find that multiple manufacturers leave all blocks unprotected, making them vulnerable to our attack. In particular, we observe that three vendors provide UDIMMs without SPD write protection for at least one of their product lines.

For approximating how common affected DIMMs are in practice, we analyse the share of the corresponding manufacturers in the memory market. Sources suggest that Corsair's total market share across all memory types was 17.4% [54] in 2025, with further sources suggesting this share to be significantly higher in the high-performance and gaming memory segments, at 55% and 72%, respectively [51, 55]. ADATA's market share is estimated to be 5% on the overall memory market [57]. Unfortunately, we were unable to find any reliable figures for G.Skill, but note that the manufacturer observes high popularity in the gaming and overclocking communities.

We conclude that *Download More RAM* is highly applicable and affects a wide range of system configurations on the consumer market. However, we want to note that we have not exhaustively tested all ADATA, G.Skill, and Corsair product lines and different product lines may have different write protection configurations, as observed for G.Skill.

7.2 Case Studies

We present four end-to-end attack case studies building on top of *Download More RAM*, showing example threats posed by the attack to endpoint systems. We then should how the attack can be executed as a one-click script without user interaction. We implemented all case studies on a DDR4 AMD system running Windows 11 25H2. Specifically, we use a ROG Strix B450-F motherboard with a Ryzen 7 2700 CPU, and a single 8GB DIMM without SPD write protection.

Disabling PPL protected AV/EDR. When considering attacks from privilege, such as ransomware or rootkits, a common step in many real-world attack chains is disabling the anti-malware solution present on the target. This is also a common use for BYOVD attacks, which have seen a rise in recent years [48]. To stop malware with admin privileges from disabling important processes, Microsoft introduced Process Protection Light (PPL) [41]. First included in Windows 8.1, PPL lets binaries signed by Microsoft run at one of seven protection levels, protecting a process from read, write, and termination requests issued by processes at lower levels. Typically Anti-virus and EDR processes run at level 3, DRM solutions at level 1 and critical Windows components at level 6. This means that even an admin level process cannot stop a PPL protected anti-virus scan, which can only be stopped by processes running at higher levels, the kernel, or by itself.

Hence, to allow user-controlled process termination, solutions running on higher protection levels often introduce secondary authentication channels to verify that termination requests are legitimate. For instance, Sophos EDR, the target of our case study, requires a "tamper protection password" to be obtained from its management web site before it can be stopped by a user [53].

The most naïve way to disable PPL would be to find and adjust the PPL protection flag directly in memory. However, we note that this would require manually locating the process in aliased memory limiting the portability of a potential attack. Instead, we leverage *Download More RAM* to to disable the vulnerable driver blocklist and revive the "*TrueSight*" attack first reported in late 2024 [45]. This BYOVD attack used a signed driver from the *RogueKiller* anti-malware tool that, ironically, can also disable EDR solutions.

First, we create polymorphic versions of the *TrueSight* driver by editing its checksum values in the PE header. This gives 2^{32} possible combinations over the 4 checksum bytes. This changes the driver's hash without affecting its signature,

so that the certificate still matches while hash based detection fails. Due to the changes in the driver integrity checks introduced by *Download More RAM*, we can now load the *TrueSight* driver. Then, we use our own modified build of the *TrueSight Killer demo* usermode program [48] to send IOCTLS to the loaded driver and disable Windows Defender virus and threat protection. Under normal operation, this application is blocked by Windows Defender when attempting to load its driver. However, our modified demo code relies on the driver being loaded already via *Download More RAM*, decoupling driver loading from attack execution. We provide a video demonstrating the full attack chain for this scenario with our artifact.[9]

To understand if *Download More RAM* also applies to EDR solutions beyond Windows Defender, we extend this attack to Sophos Intercept X. While Intercept X does not prevent aliased-based patching of code integrity and loading of the polymorphic driver, their anti-malware solution successfully blocked our userland TrueSight attack code. However, given that we can successfully load arbitrary vulnerable drivers, we argue that core protection guaranties of the EDR are already bypassed. Additionally, variants of Intercept X do not bundle the anti-malware package detecting the TrueSight usermode program, and instead rely on Windows Defender. In this scenario, we are able to disable the full EDR monitoring service without further modification to the attack.

Reading Memory of VBS protected processes. A newer and novel feature of Virtualization Based Security are Microsoft's VBS Enclaves [40]. This framework allows third parties with a valid code signing certificate to write dll files which run in the Isolated User Mode (IUM) and gain the protections offered by this isolation. These are loaded and called by a host application running in the regular user mode and are heavily restricted in functionality. Since VBS enclaves can access their host's memory, but not vice versa, they allow for secure key-value stores and other use cases relying on decrypted secrets never leaving VTL-1.

To demonstrate attacks against VBS enclaves, we write our own and load it via Windows test mode. The enclave implements a simple XOR decryption routine, which returns success every-time. The applications predefined variables are XORed to create a secret key, which returns success if it matches a stored value.

To break both confidentiality and integrity provided by the VBS enclave, we scan the aliased memory, leak the enclave's code and data, and patch it to perform a logical AND instead of the XOR. After patching, we observe the application returning a non-successful operation, confirming that *Download More RAM* can modify the logic of VBS enclaves.

[9] https://github.com/SamCollins1327/Download-More-RAM/blob/main/DownloadMoreRAM_Demo2.mp4

Bypassing Game Anti-Cheats. Recent work has shown the significant size of the game cheat market, as well as the close ties between anti-cheat and EDR solutions [10, 11]. Our attacker model (privileged user on Windows systems) closely aligns with the attacker model anti-cheat solutions ought to defend against. Hence, we leverage *Download More RAM* patch running anti-cheat systems in memory. This would allow to disable key functionality and directly deliver cheats to running game processes. We tested the *Download More RAM* against three prominent kernel-level anti-cheat solutions: Easy Anti-Cheat, BattlEye, and Riot Vanguard. No system prevented us from modifying the kernel and secure kernel code integrity checks. However, we note that BattlEye does block the used *winpmem* driver from loading, requiring the attack to be executed before a target game is started. As proof-of-concept, we tested patching Riot Vanguard's *vgk.sys* driver by disabling an example piece of their detection mechanism: a hook that the driver installs. We are able to disable this hook without apparent detection[10].

While we do not explore the possibility of a game cheat running in VTL1, there is evidence of cheats already running in custom hypervisors; we see this as a realistic use of *Download More RAM* in the wild against anti-cheat systems [10].

Overcoming Mobile Device Management (MDM). Last, we investigate the effects of *Download More RAM* on Mobile Device Management policies as commonly encountered in large firms, corporations, and, most recently, academic institutions. To avoid tampering with monitored live systems without explicit consent, we set up our test machine with a group policy mimicking that of our institution's Mobile Device Management, locking the anti-virus on and enforcing all available Device Security measures. Despite this measures, we find that *Download More RAM* can still disable Windows Defender and then gain arbitrary access to system memory. We note that the previous two steps must be executed in this order, as Windows Defender would otherwise prevent the *RTCore* software used in our proof-of-concept from running.

One-click AV bypass. The attacks above require the user to log in to their system between the memory aliasing and the exploitation steps; this could limit the attacks effectiveness as part of large attack campaigns. To show that *Download More RAM* could be automated we have developed a one-click script that aliases the memory, creates a new account, sets a start-up script for the account and sets it to auto log on. The script then reboots into that account without user interaction. The start-up script kills the PPL protected anti-virus using the TrueSight driver, as above, and runs an executable payload.

The PowerShell script creates a new account because we do not assume that the attacker would know the current user's

[10] While both Riot and Epic games responded positively to our disclosure, neither are shipping mitigations until there is proof of our attack in the wild. As such we do not release concept code for these attacks.

password. Before the reboot, the script creates the files for the account and sets a startup script to run with admin privileges. The script also sets the new account to have accepted the Windows privacy terms to avoid a pop up box that would block the reboot. The only requirement is that the script is run with admin privileges, which could be done with a remote connection to the machine. Once started no user interaction is required. The full script is available on our website.

This shows that *Download More RAM* could be used in automated, wide-scale attack campaigns, and the primary constraint on the attack is the prevalence of DIMMs with writeable SPD configurations.

8 Mitigations

Mitgations against *Download More RAM* can be implemented at various levels and require cooperation by DIMM manufacturers and operating system vendors, while users may leverage existing BIOS features as a temporary workaround.

DIMM-based defenses. Ensuring that new DIMMs use SPD write protection would prevent software based memory aliasing. For already deployed chips, we recommend to retroactively enable write protection. This can either be done by writing the raw I^2C commands, or leveraging vendor-provided tooling. However, DIMMs may block a write protection override, limiting the applicability of this defense.

BIOS Mitigations. A subset of motherboards provide a setting to disallow SPD writes via the BIOS. Affected users leveraging according platforms can use this to mitigate *Download More RAM* until further mitigations are available.

OS-Level Mitigations. OSs can either aim to detect the presence of aliased memory or to limit their potential for abuse. For instance, the OS could scan the memory during early boot for aliases or explicitly disallow DIMMs without SPD write protection enabled on secure systems. To limit the impact of aliased memory, we recommend to remove critical boot parameters such as `removememory` for secure boot compliant systems. Without these boot parameters, the OS will always use memory from the lower and upper half, and while negatively impacting system stability, it would also hinder *Download More RAM*-style attacks.

Microsoft Mitigations. Official mitigation by Microsoft was pushed as part of the April 14, 2026, Windows security update. While details of the mitigation are unavailable, we observed two main behaviour changes to the OS. First, the `removememory` boot parameter is no longer available on systems running Secure Boot. Unlike other parameters which are deleted, `removememory` is now silently ignored. Second, the suite of VBS defences, including HVCI and Credential Guard, now run on systems without secure boot enabled. Previously, secure boot was a strict requirement for VBS defences.

As a result, systems without secure boot are still affected by *Download More RAM*, while our attack, in its current form, is mitigated on systems running under secure boot. However, we note that we use `removememory` only to stabilise Windows when running with aliased memory. The rest of the attack chain is not affected by the mitigation and alternative ways for stabilising the OS would re-enable *Download More RAM*, even on systems with secure boot enabled.

9 Discussion

Robustness and Generality of Attack. The full *Download More RAM* attack leverages multiple distinct steps in combination, whereas each of these steps could introduce potential mitigations. However, it is important to note that each step of our attack may be substituted for alternate techniques and the individual attack building blocks can be reused in different attack scenarios. For instance, on a system with locked DIMM SPD chips, attackers with physical access may use traditional BadRAM techniques to create memory aliases. Similarly, the presented technique for disabling Windows code integrity verification remains practical under any arbitrary write primitive.

Vulnerable Drivers versus Unsigned Drivers. The primitives provided by *Download More RAM* would also allow patching of integrity checks verifying whether a driver is signed by a legitimate authority. While this would allow to load arbitrary, self-written drivers, we opted for re-enabling previously blocked drivers for two reasons: First, vulnerable drivers already allow for a wide range of attacks and are well-documented by the malware and cheat development communities, with readily available tools relying on known but blocked drivers. Second, allowing the reintroduction of vulnerable drivers requires less modifications to the code integrity modules, leading to a simpler and less invasive approach.

Admin-privileges on Windows. Windows OS is commonly used in enterprise environments, as such, local admin on a Windows machine does not equate to complete control of that machine. Group policy and mobile device management can be set up to restrict the capabilities of an admin, for example not allowing them to change security settings. This can also be seen in third party EDR systems such as Sophos Intercept-X Endpoint which we test. A local admin cannot disable the EDR without the master password from the network administrator. This is why admin access can be considered in scope in many windows attacks on enterprise systems, an admin cannot simply disable the security features as might be expected on a Linux endpoint.

RAM Disk Stability Using a RAMDISK to read/write live memory introduces significant side effects. The two main effects being (a) replay caused by re-writing data around the target (b) corruption of important data from the filesystem headers. To the best of our knowledge, the Primo Ramdisk utility does not allow for the creation of a virtual drive without a filesystem. We found allocating large RAM disks, 100MB and larger, causes a near 100% immediate crash rate on our test setup. In our final attack we use a 3MB RAM disk, having also found success up to 7MB.

JEDEC compliance. BadRAM [13] suggests that attacks similar to *Download More RAM* may be possible on popular consumer DIMMs which fail to comply with JEDEC standards. This was not further investigated in prior work, we demonstrate the first purely software aliasing attack, and survey the DIMM market for affected models.

10 Related Work

Windows Kernel Security. Most prior studies focus on vulnerability discovery via fuzzing (e.g., [7, 49, 50, 59]) and symbolic execution (e.g., [5, 6, 17, 31]). Beyond this, only a limited amount of work discusses exploitation techniques for the windows kernel [30]. Gruss et al. presented a prefetch side-channel based attack to bypass KASLR on Windows 10 [16], Shafir demonstrates an I/O ring technique to achieve an arbitrary kernel read/write primitive, Jagt et al. investigate recent software vulnerabilities in Windows 11's secure kernel [22], and Monzani et al. analyse BYOVD techniques as used by malware [42]. In contrast to this, *Download More RAM* leverages a flaw in insecure hardware and we present novel ways to bypass DSE and HVCI, independent of specific software vulnerabilities.

Rowhammer. Rowhammer attacks break memory isolation by inducing targeted bit flips in memory from software [29]. These attacks have been widely studied [43,44], and variations allow to even read data from memory [32]. Recent examples of Rowhammer additionally bypass mitigations in ECC [28] and DDR5 memory [35]. While these attacks allow flipping and reading individual bits in memory, *Download More RAM* enables an attacker to read and rewrite large chunks of memory, ultimately leading to an arbitrary read/write primitive.

Cold Boot Attacks. Cold Boot Attacks enable RAM readout by physically removing memory and reading its contents [4, 15, 18]. Follow up work looked extensively at unscrambling physical memory [60] and showed applications beyond desktop systems, including smartphones [19, 56] and embedded devices [58]. *Download More RAM* directly leverages the insights of these attacks to unscramble memory images, but does not rely on physical access.

BadRAM attacks. Recently, BadRAM [13] leveraged aliased memory to compromise AMD SEV-SNP enclaves, with later research by Louka et al. [34] extending the attack to break RISC-V PMP-based isolation. These papers assume that the attacker starts with control of the Linux kernel, an assumption that does not work for Windows, with its strong kernel protections. While both past works leverage physically tampered SPD EEPROMs, they speculate on the possibility of software-only attacks, with the latter one simulating disabled SPD write protection after manual unlocking. *Download More RAM* confirms this speculation by demonstrating the first attack without any hardware modifications and without requiring kernel-level access for the attacker.

Interposer-based attacks. Memory interposers enable a wide range of attacks and provide a promising research primitive. For instance, Interposers as proposed by Hopkins et al [20] and Lee et al. [33] allow snooping of memory being read, while more recent work leverage interposers for facilitating Rowhammer research [9, 14, 23].

De Meulemeester et al. propose a low-cost interposer enabling BadRAM attacks without SPD modifications [12]. This work requires the RAM to be physically removed from the system in order to insert the interposer. In comparison *Download More RAM* does not require specialised hardware, and can be mounted solely from software against affected DIMMs.

11 Conclusion

We presented *Download More RAM*, the first memory aliasing attack conducted purely from software, against Windows endpoint systems. We show how systems with vulnerable DIMMs can be stably configured to run our attack, allowing access to all areas of physical memory. We investigate the Windows security ecosystem and identify and reverse engineer key code integrity checks. Then, using our aliased memory access, we disable them. After bypassing the strong isolation guarantees of Windows Virtualisation Based Security, and patching Hypervisor Enforced Code Integrity, we deploy attacks assumed mitigated, including known attacks disabling Anti Virus and EDR solutions.

Acknowledgements

We would like to acknowledge the generous support of the Paul and Yuanbi Ramsay Research Fund at the University of Birmingham, which funded much of the test equipment and made this work possible. David Oswald's time and some equipment was supported by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/X03738X/1 and EP/X03738X/2.

Ethical Considerations

Stakeholders. This paper presents an attack leveraging features in commodity DIMMs to bypass Windows OS security, potentially affecting millions of users alongside third-party vendors whose products operate on Windows.

Thus, we identify the following five stakeholders impacted by the research carried out in this paper:

1. *DIMM manufacturers*, as their hardware platforms allow carrying out the presented attack.
2. *Microsoft*, as core defences of the Windows operating system are analysed and bypassed.
3. *Third-party vendors* whose software relies on Microsoft's OS-level defenses.
4. *End users*, as their systems could be put at risk.
5. *Malicious actors* who could abuse the techniques described in this paper to cause harm.

Ethical Principles & Potential Harms. For our research, we applied the ethical principles as laid out in the Menlo report [3]. In particular, we consider the four guiding principles in the context of our work as follows:

1. *Respect for Persons.* Considering all stakeholders, our study does not involve human subjects, ensuring that the respect for individual persons is retained.
2. *Beneficence.* This paper describes an attack and public knowledge about it allows mitigations on affected systems and the creation of more secure systems in the future. However, the research in this paper could also lead to reputational damage and financial losses for stakeholder groups 1–3. For end users, the outcomes of this research could cause severe privacy and security risks if unmitigated and abused by malicious actors. We apply the mitigations described below to minimize the risks for benign stakeholders.
3. *Justice.* In the context of this research, justice is relevant to stakeholder group 1 and 3. To ensure fairness, we investigated whether different DIMM manufacturers are affected by *Download More RAM*. Similar, we also selected a range of stakeholders in group 3 to demonstrate that the presented issue is a systematic problem, rather than an oversight by a particular vendor.
4. *Respect for Law and Public Interest.* While carrying out the research, we ensured that all experiments are executed in controlled environment, without sensitive user data deployed.

Mitigations. Throughout our research, disclosing our findings to affected parties has given highest priority. Following coordinated disclosure practices, all vendors affected by the findings of this paper have been notified with technical details and our intention to publish before submission of this paper.

All vendors acknowledged our findings and Microsoft issued CVE-2026-23670, allowing to inform downstream vendors about our findings. We further worked together with key stakeholders (i.e., Microsoft & Corsair) to implement technical mitigations against the presented attack, and Microsoft released mitigations before publication of this work.

Decision to Conduct and Publish Research. The benefits of conducting and publicizing the research leading to the *Download More RAM* attack outweighs the potential negative impact. Public knowledge about the attack allows vendors to fix the underlying vulnerabilities and develop mitigations where fixes are not attainable. Additionally, end users can verify whether their systems are vulnerable and leverage the adhoc mitigation described in this paper.

However, we do acknowledge that publishing this research informs malicious actors about the underlying vulnerabilities and attack techniques. Nonetheless, the alternative of not publishing the research would keep systems at risk, as (a) malicious actors may learn about the individual attack primitives independently about the paper at any time and (b) end users would be unable to understand and test whether they are vulnerable to according attacks.

Open Science

Our artifact is available at:
https://doi.org/10.5281/zenodo.20331553

The artifact contains *(i)* precompiled tools to reproduce *Download More RAM* on affected systems, *(ii)* source code and proprietary files for the different tools, *(iii)* Screenshots supporting the findings of our paper, *(iv)* a video demonstrating an end-to-end *Download More RAM* attack, and *(v)* a README file detailing the steps to reproduce the attack.

Part of the included tools were not developed by us. We distribute modified versions of licensed software under the respective base licenses and summarise our changes in the README file. All credit to any open source tool we use in our attack chain goes to their original creators.

References

[1] Adnan Akhunzada, Mehdi Sookhak, Nor Badrul Anuar, Abdullah Gani, Ejaz Ahmed, Muhammad Shiraz, Steven Furnell, Amir Hayat, and Muhammad Khurram Khan. Man-at-the-end attacks: Analysis, taxonomy, hu-

man aspects, motivation and future directions. *Journal of Network and Computer Applications*, 48, 2015.

[2] Daniel G Arce. Malware and market share. *Journal of Cybersecurity*, 4(1):tyy010, 12 2018. arXiv: https://academic.oup.com/cybersecurity/article-pdf/4/1/tyy010/27285055/tyy010.pdf, doi:10.1093/cybsec/tyy010.

[3] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. The Menlo Report. *IEEE Security & Privacy*, 10(2), 2012.

[4] Johannes Bauer, Michael Gruhn, and Felix C. Freiling. Lest we forget: Cold-boot attacks on scrambled ddr3 memory. *Digital Investigation*, 16:S65–S74, 2016. DFRWS 2016 Europe. URL: https://www.sciencedirect.com/science/article/pii/S1742287616300032, doi:10.1016/j.diin.2016.01.009.

[5] Mengchen Cao, Xiantong Hou, Tao Wang, Hunter Qu, Yajin Zhou, Xiaolong Bai, and Fuwei Wang. Different is good: Detecting the use of uninitialized variables through differential replay. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[6] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. S2E: A platform for in-vivo multi-path analysis of software systems. *Acm Sigplan Notices*, 46(3), 2011.

[7] Jaeseung Choi, Kangsu Kim, Daejin Lee, and Sang Kil Cha. NtFuzz: Enabling Type-Aware Kernel Fuzzing on Windows with Static Binary Analysis. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021.

[8] Mike Cohen. Winpmem – a physical memory acquisition tool. Accessed: 2026-02-05. URL: https://github.com/Velocidex/WinPmem.

[9] Lucian Cojocar, Kevin Loughlin, Stefan Saroiu, Baris Kasikci, and Alec Wolman. mfit: A bump-in-the-wire tool for plug-and-play analysis of rowhammer susceptibility factors. *Technical Report–Microsoft Research*, 2021.

[10] Sam Collins, Marius Muench, and Tom Chothia. Watching the watchers: Exploring and testing defenses of anti-cheat systems. Black Hat USA, 2025.

[11] Sam Collins, Alex Poulopoulos, Marius Muench, and Tom Chothia. Anti-cheat: Attacks and the effectiveness of client-side defences. In *Proceedings of the 2024 Workshop on Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks*, 2024.

[12] Jesse De Meulemeester, David Oswald, Ingrid Verbauwhede, and Jo Van Bulck. Battering RAM: Low-cost interposer attacks on confidential computing via dynamic memory aliasing. In *IEEE Symposium on Security and Privacy (S&P)*, 2026.

[13] Jesse De Meulemeester, Luca Wilke, David Oswald, Thomas Eisenbarth, Ingrid Verbauwhede, and Jo Van Bulck. BadRAM: Practical memory aliasing attacks on trusted execution environments. In *IEEE Symposium on Security and Privacy (S&P)*, 2025.

[14] Stefan Gloor, Patrick Jattke, and Kaveh Razavi. REFault: A Fault Injection Platform for Rowhammer Research on DDR5 Memory. In *Microarchitecture Security Conference (uASC)*, 2025.

[15] Michael Gruhn and Tilo Müller. On the practicability of cold boot attacks. In *Availability, Reliability and Security (ARES)*, 2013.

[16] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[17] Rajat Gupta, Lukas Patrick Dresel, Noah Spahn, Giovanni Vigna, Christopher Kruegel, and Taesoo Kim. POPKORN: Popping Windows kernel drivers at scale. In *Annual Computer Security Applications Conference (ACSAC)*, 2022.

[18] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM (CACM)*, 52(5), 2009.

[19] Christian Hilgers, Holger Macht, Tilo Müller, and Michael Spreitzenbarth. Post-mortem memory analysis of cold-booted android devices. In *Eighth International Conference on IT Security Incident Management & IT Forensics*, 2014.

[20] Bradley Hopkins, John Shield, and Chris North. Redirecting dram memory pages: Examining the threat of system memory hardware trojans. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2016.

[21] integralfx. DDR4 XMP Editor. Accessed: 2026-02-04. URL: DDR4XMPEditor.

[22] Jonathan Jagt, Daan Keuper-Computest, Erik Poll, and Xavier de Carné de Carnavalet. Analysis of Windows Secure Kernel security bugs. Master's thesis, Computest & Radboud University, 2025.

[23] Patrick Jattke, Michele Marazzi, Flavien Solt, Max Wipfli, Stefan Gloor, and Kaveh Razavi. McSee: Evaluating Advanced Rowhammer Attacks and Defenses via Automated DRAM Traffic Analysis. In *USENIX Security Symposium*, 2025.

[24] JEDEC. Definitions of the EE1004-v 4 kbit serial presence detect (SPD) EEPROM and TSE2004av 4 kbit SPD EEPROM with temperature sensor (TS) for memory module applications. Standard 21-C, Section 4.1.6, JEDEC, May 2022.

[25] JEDEC. SPD5118 hub and serial presence detect device standard. Standard JESD300-5B.01, JEDEC, May 2023.

[26] JEDEC Solid State Technology Association. Definitions of the EE1004-v 4 kbit serial presence detect (SPD) EEPROM and TSE2004av 4 kbit SPD EEPROM with temperature sensor (TS) for memory module applications. URL: https://www.jedec.org/standards-documents/docs/spd416.

[27] Jeff. RWEverything - Read & Write Everything. Accessed: 2026-02-04. URL: https://rweverything.com/.

[28] Nureddin Kamadan, Walter Wang, Stephan van Schaik, Christina Garman, Daniel Genkin, and Yuval Yarom. ECC.fail: Mounting Rowhammer Attacks on DDR4 Servers with ECC Memory. In *USENIX Security Symposium*, 2025.

[29] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *International Symposium on Computer Architecture (ISCA)*, 2014.

[30] Anil Kurmus, Andrea Mambretti, Alessandro Sorniotti, Vincent Lenders, Damian Pfammatter, and Bernhard Tellenbach. SoK: Automating Kernel Vulnerability Discovery and Exploit Generation. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2025.

[31] Volodymyr Kuznetsov, Vitaly Chipounov, and George Candea. Testing Closed-Source Binary Device Drivers with DDT. In *USENIX Annual Technical Conference (ATC)*, 2010.

[32] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. Rambleed: Reading bits in memory without accessing them. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[33] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia-Che Tsai, and Raluca Ada Popa. An off-chip attack on hardware enclaves via the memory bus. In *USENIX Security Symposium*, 2020.

[34] Antonis Louka, Jesse De Meulemeester, Steven Keuchel, Ingrid Verbauwhede, and Jo Van Bulck. Physical memory please: Practical memory-aliasing attacks on risc-v pmp. In *Microarchitecture Security Conference (uASC)*, 2026.

[35] Diego Meyer, Patrick Jattke, Michele Marazzi, Salman Qazi, Daniel Moghimi, and Kaveh Razavi. Phoenix: Rowhammer attacks on ddr5 with self-correcting synchronization. In *IEEE Symposium on Security and Privacy (S&P)*, 2026.

[36] Microsoft Corporation. Smart app control. Microsoft Learn. Accessed: 2026-02-04. URL: https://learn.microsoft.com/en-us/windows/apps/develop/smart-app-control/overview.

[37] Microsoft Corporation. Enable virtualization-based protection of code integrity, 2025. Last updated 15 August 2025; accessed August 10, 2026. URL: https://learn.microsoft.com/en-us/windows/security/hardware-security/enable-virtualization-based-protection-of-code-integrity?tabs=security.

[38] Microsoft Docs. *Kernel-Mode Code Signing Requirements–Windows Drivers (Windows Vista and Later)*. Microsoft, 2024. Accessed: 2026-01-21. URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-requirements--windows-vista-and-later-.

[39] Microsoft Learn. *Virtualization-based Security (VBS)*. Microsoft, 2025. Accessed: 2026-01-21. URL: https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-vbs.

[40] Microsoft Learn. *Virtualization-based security (VBS) enclaves overview*. Microsoft, 2025. Accessed: 2026-01-21. URL: https://learn.microsoft.com/en-us/windows/win32/trusted-execution/vbs-enclaves.

[41] Microsoft Learn. *Protecting Anti-Malware Services*. Microsoft, 2026. Accessed: 2026-01-21. URL: https://learn.microsoft.com/en-us/windows/win32/services/protecting-anti-malware-services-.

[42] Andrea Monzani, Antonio Parata, Andrea Oliveri, Simone Aonzo, Davide Balzarotti, and Andrea Lanzi. Unveiling byovd threats: Malware's use and abuse of kernel drivers. In *Symposium on Network and Distributed System Security (NDSS)*, 2026.

[43] Onur Mutlu and Jeremie S Kim. Rowhammer: A retrospective. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8), 2019.

[44] Onur Mutlu, Ataberk Olgun, and A Giray Yağlıkçı. Fundamentally understanding and solving rowhammer. In *Asia and South Pacific Design Automation Conference (ASDAC)*, 2023.

[45] Check Point Research. Silent killers: Unmasking a large-scale legacy driver exploitation campaign, February 2025. Accessed: 2026-02-03. URL: https://research.checkpoint.com/2025/large-scale-exploitation-of-legacy-driver.

[46] Romex Software. Primo Ramdisk. https://www.romexsoftware.com/en-us/primo-ramdisk/, 2026. Product page for Primo Ramdisk software.

[47] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals, Part 1*. Microsoft Press, 7 edition, 2017.

[48] Maor Sabag. TrueSightKiller: CPP AV/EDR killer. GitHub repository, November 2023. URL: https://github.com/MaorSabag/TrueSightKiller.

[49] Sergej Schumilo, Cornelius Aschermann, Robert Gawlik, Sebastian Schinzel, and Thorsten Holz. kAFL: Hardware-assisted feedback fuzzing for os kernels. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 167–182, Vancouver, BC, August 2017. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo.

[50] Florian Schweins, Moritz Schloegel, Moritz Bley, Nico Schiller, and Thorsten Holz. VMIGEN: Utilizing Virtual Machine Introspection for Fuzzing Complex Closed-Source Targets. In *Annual Computer Security Applications Conference (ACSAC)*, 2025.

[51] Nikolaos Sismanis. Corsair gaming stock (nasdaq:crsr): Weak profits to quash further gains. URL: https://www.tipranks.com/news/article/corsair-gaming-stock-nasdaqcrsr-weak-profits-to-quash-further-gains.

[52] Softpedia. Thaiphoon burner. https://www.softpedia.com/get/Tweak/Memory-Tweak/Thaiphoon-Burner.shtml, September 2023.

[53] Sophos. Tamper protection, Nov 14 2025. Accessed: 2026-02-03. URL: https://docs.sophos.com/central/customer/help/en-us/index.html?contextId=tamper-protection.

[54] Statista. Memory manufacturer market share worldwide. https://www.statista.com/statistics/1465721/memory-manufacturer-market-share-worldwide/, 2025. Accessed: 2026-02-02.

[55] Caitlin Syho. Corsair gaming inc. (nasdaq: Crsr) — build it beautiful. https://westpeakresearch.com/CRSR_Caitlin_Syho.pdf, March 2022. WestPeak Research analyst report. Accessed: 2026-02-02.

[56] Benjamin Taubmann, Manuel Huber, Sascha Wessel, Lukas Heim, Hans Peter Reiser, and Georg Sigl. A lightweight framework for cold boot based forensics on mobile devices. In *Availability, Reliability and Security (ARES)*, 2015.

[57] Tom's Hardware. Adata chairman says ai datacenters are gobbling up hard drives, ssds, and dram alike — insatiable upstream demand could soon lead to consumer shortages. URL: https://www.tomshardware.com/tech-industry/big-tech/adata-chairman-says-ai-datacenters-are-gobbling-up-hard-drives-ssds-and-dram-alike-insatiable-upstream-demand-could-soon-lead-to-consumer-shortages.

[58] Yuanzhe Wu, Grant Skipper, and Ang Cui. Cryomechanical ram content extraction against modern embedded systems. 2023.

[59] Jiacheng Xu, Xuhong Zhang, Shouling Ji, Yuan Tian, Binbin Zhao, Qinying Wang, Peng Cheng, and Jiming Chen. MOCK: Optimizing kernel fuzzing mutation with context-aware dependency. In *Symposium on Network and Distributed System Security (NDSS)*, 2024.

[60] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd Austin. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

Appendix

A Prototyping *Download More RAM* attacks in Windows Test Mode

Windows test mode is a boot parameter which allows the operating system to run driver or system files which are not properly signed by Microsoft. Test mode allows developers to test new drivers quickly, skipping the usual signing requirement, and removing the risk of buggy mid-development code from being signed. Enabling test mode allows us to load custom drivers, which in turn grants access to more powerful kernel-mode capabilities. This enables more direct interaction with low-level resources. While this setup is unrealistic in an attack scenario, it allows to better explore a system running in an aliased configuration, discover Windows components that might be vulnerable, and check which security boundaries were broken by the presence of aliased memory.

Confirming access to aliased memory. While developing *Download More RAM*, we created a custom kernel mode driver which uses `MmMapIoSpace` to allocate a buffer encompassing the entirety of our aliased memory region and write its memory to a file. The allocation of this space succeeded in testing, demonstrating that the `removememory` boot parameter does not limit access from the kernel. We also used this driver to confirm that we are able to write into the aliased memory using the same buffer.

Identifying tools for reading aliased memory. Many forensic applications can dump physical memory on Windows in a conventional setting. However, during initial investigation, we failed to identify any tool capable of accessing the aliased memory locked by our boot configuration. We wanted a tool whose driver used the `MmMapIoSapce` kernel function to allocate space, as we found this to be effective in test mode configurations.

Because each forensic application initially tested used its own driver, we decided to try and develop a BYOVD tool using one of these. To find a driver using `MmMapIoSpace` internally, we disassembled candidates in Ida Pro, and inspected invoked windows kernel functions searching for `MmMapIoSpace`. We found the tool winpmem [8], and its accompanying driver `winpmem_x64.sys`, were ideal candidates.