On the Creation of a Secure Key Enclave via the Use of Memory Isolation in Systems Management Mode

James Andrew Sutherland, Natalie Coull & Robert Ian Ferguson
Division of Cybersecurity
Abertay University
Dundee, UK
email: {j.sutherland, n.coull, ian.ferguson}@abertay.ac.uk

***Abstract* — One of the challenges of modern cloud computer security is how to isolate or contain data and applications in a variety of ways, while still allowing sharing where desirable. Hardware-based attacks such as RowHammer and Spectre have demonstrated the need to safeguard the cryptographic operations and keys from tampering upon which so much current security technology depends. This paper describes research into security mechanisms for protecting sensitive areas of memory from tampering or intrusion using the facilities of Systems Management Mode. The work focuses on the creation of a small, dedicated area of memory in which to perform cryptographic operations, isolated from the rest of the system. The approach has been experimentally validated by a case study involving the creation of a secure webserver whose encryption key is protected using this approach such that even an intruder with full Administrator level access cannot extract the key.**

Keywords- key-enclave; hardware security; system-management mode.

I. INTRODUCTION

Computer security is largely concerned with erecting boundaries between entities: users, privilege levels, processes. Wherever a resource crosses a boundary, it creates the potential for compromise, either through passive information leakage (as in the case of timing attacks, where the exact details of how long an operation takes inadvertently discloses some information) or the potential for active tampering (as in RowHammer [1], where writing to one memory location indirectly affects another through non-obvious electrical coupling between parts of a memory chip).

A. Motivation

Attacks based upon covert channels and side channels depend on unexpected interactions; RowHammer for example, can be used to achieve privilege escalation via a previously-unexpected interaction between physically proximate memory components [2] . Since there was no correlation between physical and virtual addresses, as different processes and the kernel would commingle pages arbitrarily, low-privilege pages could easily be found which happened to be adjacent to highly sensitive system ones, allowing tampering. The same applies between virtual machines and hypervisor control structures. As detailed later, the more coarse-grained the sharing gets, the more limited the avenues of attack become, though any level of shared caching can be an avenue of attack [3].

As encryption keys are typically stored in RAM, a successful compromise of a system via techniques such as these can reveal those keys used to protect data at rest on the system, e.g., full-disk encryption, and data in transit to/from the system, e.g., via an SSL connection.

The ability to improve segregation of memory to securely store keys etc. separately from less sensitive data has previously required a system to have dedicated features, e.g., Intel's SGX integrated with the processor. The consequences of an attack that compromises such facilities can be widespread: In the case of SGX, this protection was defeated in 2018 via side-channel attack [4], forcing Intel to update SGX's deployment mechanism to be able to check whether the Spectre [5] attacks were properly mitigated on the target hardware.

B. An alternative approach to creating an enclave

The current generation of Intel processor architectures have a feature called Systems Management Mode (SMM) which can be used during the boot process to create an area of RAM (SMRAM), which is subsequently 'locked' and thus rendered inaccessible/unusable by 'userland' code. This offers the possibility of creating a secure memory enclave for the storage of cryptographic keys and the code which manipulates them (negotiation, verification etc.) The locked area can only be accessed by returning to SMM mode which automatically executes the code that has been securely locked in that area. This fact led to the following research hypothesis for the work:

Secure isolation can be practically implemented using only the long-established Systems Management Mode mechanisms, giving better security isolation than existing techniques such as process separation.

The work described in the remainder of this paper shows how this can be used to create a secure enclave. It is worth noting that some other processor architectures, e.g., ARM, have equivalent facilities and the proposed technique for enclave creation is thus generalisable.

The material in the paper is based on the PhD thesis of the first author and is published here for the first time [6].

The remainder of the paper is structured thus: In Section II previous work on providing secure key stores is considered. This acts as a baseline for comparison with the technique presented here. Section III describes the proposed solution to this problem whilst Section IV discusses how the

 ISBN: 978-1-68558-044-5

approach was evaluated. The results of the evaluation are given in Section V. Conclusions and proposals for further developing the approach are given in Section VI.

II. BACKGROUND

The provision of cryptographic services to a system depends upon the inviolability of any stored keys. As such services form the basis of secure computing, a secure place to store them is referred to as a Trusted Computing Base (TCB). Finding a means of creating such a TCB in RAM is thus an important security problem. This section therefore reviews various attempts at organising and protecting memory, dating from early multi-tasking operating systems and the consequent need to provide process separation through to recent hardware crypto-key enclaves before going on to review the solution-space technique of System Management Mode.

A. Protecting memory

1) Memory management/virtual memory

The idea of programs sharing system resources without interfering with each other can be traced back to the MIT 'Compatible Time Sharing System' [7]. Prior to this, only one process would be executing hence the idea of 'interference' did not apply.

Modern processor architectures implement some form of virtual memory mapping [8]: the memory a user process can access at address 0x10000, for example, may be stored in any arbitrary page of physical memory, or indeed be entirely absent and filled in by the operating system when an attempt is next made to access that, known as a 'page fault'.

To reduce the overhead of loading this mapping from memory, processors generally feature Translation Lookaside Buffers (TLBs), a set of cached address mappings. (Architectures have varying approaches to this; on MIPS, the operating system explicitly populates TLB entries as needed; x86 and more recent ARM variants populate TLB entries directly within the hardware without OS involvement, while the original ARMv2 had 512 explicit memory mappings within the MEMC1 memory controller chip as Content Addressable Memory.)

A key concept in ensuring that concurrently executing programs cannot interfere with each other or access their data is that each process be allocated its own set of memory pages and be unable to access RAM outwith those bounds. Attacks such as RowHammer, Heartbleed [9] and Spectre have shown that such OS-enforced restrictions can be circumvented and thus a more secure approach is required when storing particularly sensitive information such as encryption keys.

2) RAM Encryption

TRESOR [10] demonstrated that a general-purpose computer system can be operated with almost all of its main-memory encrypted while at rest, albeit with a significant performance penalty, using a modified Linux kernel. There is some overlap with the research this paper describes: TRESOR uses the processor debug registers as an area of storage which cannot be accessed via Direct Memory Access (DMA). This was intended to protect against DMA attacks, among others, but was not successful in that respect since this cannot protect the associated code: TRESOR-Hunt [11] demonstrated a successful attack on this protection, using code injection via DMA - an attack which could not be prevented through software mechanisms alone.

TreVisor [12] extended the techniques of TRESOR to a hypervisor level in combination with techniques from BitVisor [13] to incorporate Intel VT-d (IOMMU) protection from DMA attack.

On other platforms, the ARMORED [14] project applied TRESOR techniques to the Android operating system on ARM architecture processors as a countermeasure to their own FROST [15] attack, which used a cold boot attack to retrieve information from mobile handsets running Android 4.0 despite the disk encryption employed.

3) Address Space Layout Randomisation - ASLR

Traditionally software systems (and operating systems in particular) locate certain critical pieces of information at well-known, or at least predictable, memory addresses. Having its origins in the (Linux) PaX project [16] ASLR involves varying the location of memory contents over time thus making it more difficult for an attacker to find those critical locations.

4) Swap encryption

A cold boot attack can retrieve RAM contents for a brief period after a system is shut down, but the system's virtual memory persists indefinitely after shutdown unless explicitly wiped. To avoid this, keeping that data encrypted is an idea which long predates efforts to encrypt or otherwise protect the RAM, including the encrypted swap space [17] extensions to the virtual memory (VM) system originally proposed as an enhancement of the original 4.4 BSD approach [7]. The much slower nature of disk storage meant the extra overhead of this encryption was more widely accepted early on.

B. Other approaches to key protection

The approaches outlined above are general in that they seek to prevent cross-process interference between any two processes. Given the sensitive nature of crypto-services/keys, i.e. the consequences of their compromise, work has been done specifically on preventing inappropriate access to such keys: This sub-section reviews some typical attempts to provide such an enclave.

1) Process separation

Process separation in a cryptographic context is a software system design principle that demands that all handling of keys and cryptographic operations be performed in a separate process from the 'worker' process thus relying on the properties of the OS memory management system to deny the 'worker' any access to sensitive information. Its importance to the current work that the performance of our SSM-based solution is compared with a 'process separation' solution in experiment 4b (See Section IV).

2) Process isolation

The commercial content delivery network (CDN) Cloudflare has an interesting implementation of TLS/SSL in two respects. First, they offer 'Keyless SSL' [18] in which the site's private key is handled remotely. Secondly, the

SSL/TLS handling is performed in a separate isolated instance of the Nginx web server — an example of defence in depth which ensured that when a bug was found in their HTML parsing implementation, the information disclosed could not include site private keys, unlike with the widespread Heartbleed bug in OpenSSL [19] — only a kernel or hardware level exploit could have exposed the key, not an application level one.

3) VM isolation/hypervisors

Microsoft recently released a software-only implementation of a similar approach, Credential Guard [20], in which authentication keys are held in a dedicated virtual machine running on top of the Hyper-V hypervisor platform. This way, even a kernel compromise of the main operating system is not sufficient to extract credentials for reuse: no more 'Pass The Hash' privilege escalation once a system is compromised. Only a compromise of the underlying hypervisor itself, or the hardware isolation mechanisms, would suffice: a much smaller attack surface compared to the full OS.

4) Trusted Platform Module

The primary alternative to the general approach outlined above, where enhanced security is needed compared to direct key handling without extra isolation, is to use a dedicated cryptographic hardware device. Some PCs and servers are now equipped with a Trusted Platform Module (TPM) which provides a dedicated cryptographic and storage facility, with a fixed set of algorithms, limited storage and minimal performance [21].

5) Intel Software Guard Extensions - SGX

Intel Software Guard Extensions aim to deliver similar benefits within the main processor through architectural extensions, with an encrypted area of main memory rather than one isolated by the memory controller hardware. SGX-Shield [22] reviews the main limitations of this implementation and proposes an implementation of ASLR (varying the location of memory contents to make attacks more difficult) within this enclave for additional protection from outside interference.

This isolation is a mixed blessing, providing a hiding place for less benign code as well [23], while failing to protect against variants of the Spectre attack [4]. The TaLoS project [24] has significant similarities to the final experiment in Section V, in that it seeks to protect the encryption keys and traffic over an SSL/TLS connection but using SGX rather than SMM to isolate the data in question.

C. System Management Mode (SMM)

The approach considered in this paper is based upon the System Management Mode of the x86 family of processors (see Figure 1). As its operation provides the security guarantees necessary for creating a key enclave, it is discussed here in detail.

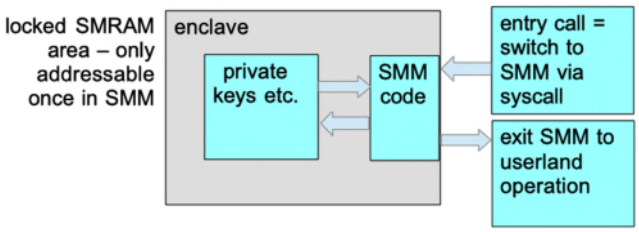

Figure 1. System Management Mode

The defining characteristic of SMM is that while the processor core is executing code in that mode, it asserts the SMIACT2 output line. This signal is interpreted by the Memory Controller Hub (MCH): when asserted, addresses are decoded differently, enabling access to the otherwise-inaccessible SMRAM area. Physically, this is just part of the main RAM, but gated by the memory controller to prevent non-SMM access. In early SMM implementations, the address used was 0xA0000, which is also used by legacy graphics support: any attempt by non-SMM code to read or write this area will access the video memory instead.

The location of SMRAM is defined by the SMBASE register, initially set to 0x30000 (192 kilobytes from the bottom of the memory space); setting the G_SMRAME control flag on the processor's SMRAMC (SMRAM control) register puts 128 kilobytes of SMRAM at a base address of 0xA0000, or 640 kilobytes, while setting T_EN (TSEG Enable) grants access to a larger area higher up. The address layout is depicted in Table I.

TABLE I. THE X86 PROCESSOR MEMORY MAP

Address	Size	Content (normal)	Content (SMM)
0xF0000	64k	BIOS ROM	
0xC0000	192k	Device ROM/Upper Memory Blocks	
0xA0000	128k	Legacy video	SMRAM
0x00000	640k	Legacy (DOS) memory	

It is important to note that SMM is not a privileged mode of execution as such, despite common references to it as 'ring -1' or 'ring -2' as if it were a more privileged alternative to ring 0 in which kernel code executes. For example, Wojtczuk and Rutkowska [25] refers to "escalation from ring 3 to SMM" — in reality, SMM code is entered in ring 0, and can transition to a reduced privilege level if desired.

In all cases, access to the SMRAM area is permitted only if the access is by the processor core (as opposed to any other peripheral), and then only if either SMIACT is asserted or the D_OPEN control bit in the system chipset is set to permit this. As a result, SMRAM has robust protection against any

 ISBN: 978-1-68558-044-5

sort of DMA attack: attempted access from the PCI bus or elsewhere is not valid at any time.

1) Bootstrapping SMM

As noted earlier, access to the dedicated area of memory SMM uses (the SMRAM) is gated by the memory controller. In order to bootstrap the SMI handler, however, it must be possible to load this memory area before the first SMI instance. This is permitted by the D_OPEN control bit in the chipset: when set, this bit permits access to SMRAM without being in SMM. After initialisation is complete, this bit should be cleared and the D_LCK (Lock) bit set, rendering all the SMM control registers read-only until the processor is reset.

This should be done very early in the system boot process by the system BIOS before activating any peripherals or executing any other code to prevent malicious code using SMM as a hiding place; older BIOS implementations often failed to secure the state properly during the boot process, leaving the way open for a variety of SMM rootkits at least as far back as 2009 [26].

2) Using SMM for security

Soon after malicious use of SMM's isolation property was demonstrated, more benign uses were found, with HyperGuard [27] in 2008, HyperCheck [28] in 2010, HyperVerify [29] in 2013 and a US patent on the concept being granted in 2014 [30].

The TrustZone-based Real-time Kernel Protection (TZ-RKP) [31] applies the same concepts to an ARM system, using ARM's TrustZone mechanism in place of SMM. (TrustZone was created later, with a 'Secure World' entered by invoking a Secure Monitor Call exception.)

The underlying concept in each case is to generate then periodically verify cryptographic hashes of critical structures or code, in HyperGuard's case, by walking the Page Tables to identify all executable pages marked for supervisor access. At the time, this was not wholly sufficient since the processor could still execute non-supervisor pages with supervisor privilege; the later development of Supervisor Mode Execution Protection (SMEP) by Intel [32] closed this loophole.

The level of privilege at which code executes in x86 Protected Mode is determined by the two least significant bits of the CS (Code Selector/Segment) register, so the code at a single address in memory may normally be executed at any privilege level without modification. This has its origins in the 80286's implementation of Protected Mode, prior to the 80386's introduction of paged virtual memory: as the two mechanisms were orthogonal, prior to SMEP a page could be user writable (ring 3) yet run at kernel privilege (ring 0).

III. PROPOSED SOLUTION

This work aims to secure a network-connected system against remote or transient physical attack, using a simple web server as the model and endeavouring to protect it against unauthorised information disclosure, in particular, disclosure of the cryptographic keys which are used to authenticate the server to clients. The keys and the code used to negotiate and verify them are protected by storing them in SMRAM as outlined in the previous section. The approach is clearly generalisable to securing the authentication material on the client end as well: client cryptographic keys, stored passwords, and payment mechanisms could also be improved. This section thus describes how a secure proof-of-concept webserver was created which uses an SMM enclave to protect the keys it uses for serving HTTPS requests.

The starting point in creating the proof-of-concept server was an OpenSSL example TLS server [33] which was linked with Google's SSL implementation: BoringSSL [34] to which was added code implementing the SMM key protection from the previous section. The server runs as a normal unprivileged application ('ring 3') under Linux and used TLS 1.2.

Key design goals for the proof-of-concept server were a minimal overhead in each transition to/from SMM, and presenting a minimal attack surface on the SMM component while enabling the application counterpart to run with minimal privileges. From the programmer's perspective, the enclave functions in a manner akin to a physical hardware device, passing messages in both directions via a page of physical memory.

A. Overall operation

Three actions are necessary at boot time:

- A public/private key pair are generated (see Section III.A.1 "Key Negotiation" below)
- The private key and the code for verifying a candidate public key are placed in the SMRAM page.
- The SMRAM is locked (using technique described in Section II.C.1)

In subsequent operations, i.e., when the webserver wishes to serve a page, there is a need to pass information to the code now locked in SMRAM. This is achieved through the use of a small (4Kb) area (known as the 'mailslot') which is accessible from both inside and outside of SMM (See Figure 2).

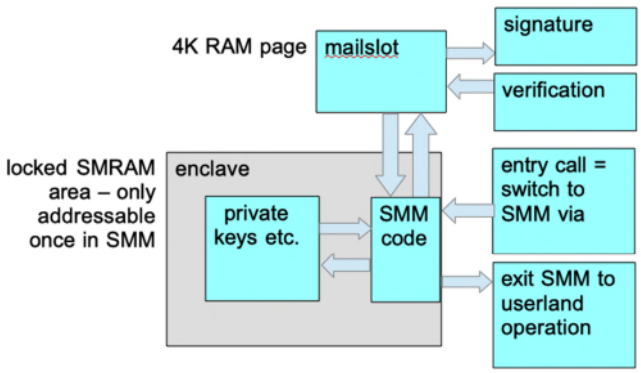

Figure 2. API/Using SMM for signature verification

Userland code inserts any public key to be verified into the mailslot, transitions into SMM (See Section III.A.2 - "Transitioning to SMM" below) which causes a jump to the code in the SMRAM. That code has access to both the mailslot RAM and the SMRAM - verifies the public key

against the private key held in SMRAM and places the result of verification(true/false) in the mailslot RAM and exits from SMM causing a return to the calling userland code.

To make use of the cryptographic enclave services, the userspace code must first allocate and lock a page of physical memory, determining the underlying physical address via the Linux /proc/self/pagemap virtual file and communicating this to the SMM enclave at initialisation time. This shared page can then be used as a mailslot for exchanging data: the userspace (ring 3) code interacts directly with the SMM cryptographic code, without transitions to/from the kernel in between.

1) Key negotiation

To protect the most sensitive data requires the construction of some sort of containment to which access from all other components is restricted or prevented — but with just enough interaction permitted to enable the intended use of the keys (or other material) in question. For an SSL/TLS web server, the sensitive data is created as a public/private key pair. As the name implies, the public part of the pair may be freely exported and shared — indeed, it is provided to every client connecting, as part of the initial protocol handshake — while the private key is never to be disclosed to anyone else. To prove the identity of the server, a Certificate Signing Request (CSR) is generated and signed using the private key; after completion of appropriate checks, a Certification Authority (either one trusted by the general public and the software they use, such as LetsEncrypt, or an internal entity such as the US Department of Defense's internal CA) usually signs that CSR to produce a certificate. Any entity can issue certificates, it is merely a matter of policy which issuers are trusted or not for any given situation; for experimental purposes, a self-issued certificate is equally suitable.

2) Transition to SMM

The process of transitioning to SMM is worth examining as it incurs an overhead and as it needs to be accomplished each time a cryptographic verification operation is required, minimising that overhead is a worthwhile goal.

Entry to SMM requires triggering an SMI (System Management Interrupt). Ordinarily, hardware interrupts cannot be triggered directly from user mode applications; first a system call would be required, to effect a transition to kernel mode ('ring 0' on x86), then the corresponding kernel code would trigger the interrupt on the application's behalf. This, however, incurs additional overhead, two mode transitions rather than one. A more efficient approach is for the application to write to the I/O address 0xb2 as explained below.

Most modern processors implement a unified hardware memory map, in which RAM and devices occupy the same address space; x86 has two distinct memory spaces, a 64 kilobyte legacy space accessed via the IN/OUT set of instructions, and a much larger space accessed via standard memory operations.

For devices mapped into the main memory space, the usual memory permissions apply: the appropriate 4 kilobyte (or larger) page could be mapped with appropriate permission bits set. The I/O space has different, fine-grained permissions: the I/O Permissions Bitmap (IOPB) within the Task State Segment (TSS) controls whether access is granted or not to any given byte within the I/O address space. On Linux, the ioperm system call may be used to enable access to any specified I/O address.

To make use of the cryptographic enclave services, the userspace code must first allocate and lock a page of physical memory, determining the underlying physical address via the Linux /proc/self/pagemap virtual file and communicating this to the SMM enclave at initialisation time. This shared page can then be used as a mailslot for exchanging data: the userspace (ring 3) code interacts directly with the SMM cryptographic code, without transitions to/from the kernel in between.

IV. EVALUATION PROCESS

In order to show that the proposed solution is practicable (and establish the hypothesis) three aspects of the proof-of-concept webserver's behaviour were evaluated: functionality, security, and performance. Functionality was demonstrated by testing with a) a number of web-browsers (Experiment 1) and b) an industry-standard test suite (Experiment 2). Security is shown by reasoning from properties of the SMM system. Performance was tested by a) examining the impact on execution time of the overhead of entering and exiting SMM through micro-benchmarking (Experiment 3) and b) comparing the time taken to serve pages i) with no key protection (Experiment 4a) ii) with 'process-separation' based key-protection (Experiment 4b) and iii) with SMM-based key protection (Experiment 4c). A summary is given in Table II below.

TABLE II. LIST OF VALIDATION EXPERIMENTS PERFORMED AND PURPOSE

Num	Experiment	Purpose
1	Use with range of browsers	Verifying basic webserver functionality
2	Qualys - SSL Labs	Verifying webserver SSL protocol compliance
3	Micro-benchmarking	Measuring the 'real-time' overhead imposed by entering and exiting SMM
4a	Comparison of webserver performance with crypto operation performed with 3 different levels of protection	Measuring the rate that pages could be served with crypto-keys handled in-process, i.e., with no protection
4b		Measuring the rate that pages could be served with crypto-keys handled in a separate process, i.e., with process-separation protection
4c		Measuring the rate that pages could be served with crypto-keys handled in SMM

As the webserver's cryptographic code is unmodified – a standard x86/x86-64 implementation of the elliptic curve algorithms – the key performance metric is the additional overhead introduced by transitions to and from SMM. For

context, this should be compared with the overhead entailed in a context switch between usermode processes (as applies where the cryptographic code is run in a separate process, as CloudFlare does in their content delivery network's edge devices) and user-kernel mode transitions particularly after implementation of the Kernel Page Table Isolation (KPTI) changes to mitigate the Spectre/Meltdown security issues. Experiments 3 and 4b quantify these.

For a better indication of the real-world performance impact, standard HTTPS benchmarking — downloading static content over encrypted connections in each configuration tested — gives indicative throughput speeds (Experiment 4).

A. Functionality

Once the HTTP-over-TLS (HTTPS) server was implemented, a variety of protocol interactions were tested. Initially, standard HTTPS clients (wget, curl, Mozilla Firefox and Google Chrome) were used to verify basic functionality (Experiment 1), and any issues encountered resolved; after this, the more comprehensive industry standard test suite - SSL Labs from Qualys [35] - was employed (Experiment 2).

B. Security

The webserver's resistance to RowHammer and Spectre attacks was analysed. While web server performance testing is a well studied and long-established field [36][37], security is more nebulous. In this context, the architecture is intended to provide isolation, and substantial literature has already studied the various possible routes to accessing SMRAM [25] — cache aliasing, Memory-Type Range Registers (MTRR) manipulation; and early BIOS implementations which neglected to enable D_LOCK timeously). It can also be verified empirically that the SMRAM-protected data/code is not exposed, even to the kernel via a scan of the Linux `/dev/mem` device, which can be configured to expose the kernel's view of the entire memory space. Since the SMM protected data has no functioning address except while the processor is executing in SMM, exploits such as Spectre cannot access this data. (Physical level attacks such as RowHammer or address line fault injection could still be effective.)

1) RowHammer

The RowHammer attack allows modification of bits in physically adjacent areas of memory, which could theoretically be used to exfiltrate information from the SMM enclave. Integrity checking would provide some protection against this, while ASLR would make such an attack almost impossible — just shifting the code and data by a small random number of bytes each time the system is booted would mean the attacker was operating blindly (able to flip some bits, but without knowledge of which instruction or piece of data is being affected), while the use of 'canary' values around the code and data would make such an attempted attack very unlikely to go undetected. Moreover, given sufficient knowledge of the memory arrangement in use, simply adding a single disused row between the SMM code and data area and memory used by the system would frustrate any RowHammer attempt: it would corrupt only that buffer space, with no effect on the SMM area.

Also, on the specific test hardware used for the majority of this experimentation, the DDR2 memory installed is much less susceptible to RowHammer attacks anyway: exploiting this generally requires DDR3 or newer, due to the smaller feature size and faster access.

A similar approach would also be effective against most direct hardware attacks, such as address line glitching: without knowing the exact address to target, a successful attack would be very much more difficult than against a system without this protection.

2) Spectre/Meltdown

The most recent memory protection attacks against vulnerable Intel and ARM processor architectures pose two potential threats against an SMM protection implementation.

Firstly, the Meltdown techniques can be used directly to extract otherwise protected data, for example from kernel buffers, by using the address of that data indirectly then observing side-effects of that operation. This is not applicable to SMM code or data, since there is no address which refers to that memory in the first place. This was empirically verified by Eclypsium[38].

Secondly, the Spectre attacks have been used against system firmware executing in SMM to bypass bounds checks (ibid.) — that issue is avoided entirely in this work by using only fixed size parameters, with no bounds checks or boundaries to be violated.

C. Performance

For the performance assessment, two approaches are used: first (Experiment 3), microbenchmarks, measuring the individual components involved in transitions to and from SMM and kernel mode in isolation ; secondly (Experiments 4a - 4c), to measure the overall performance of a web server using different isolation mechanisms, to be able to compare SMM isolation's performance overhead against versions with no isolation of key handling and one which uses process-level isolation which would protect against process level compromise, but not a root or kernel level one as SMM isolation does.

1) Experiment 3 - Microbenchmarking the mode transition cost

The experiment described here investigates the performance aspects of using SMM, detailing the performance impact of each transition to and from SMM compared to transitions to kernel space and back which is the dominant factor in the overall performance of the SMM-isolated server.

After prototyping work on the Bochs hardware simulation, a physical target system was required for performance tests. A Lenovo ThinkPad X200 was obtained and loaded with the Libreboot free software project's variant of the open-source Coreboot firmware (Libreboot), including its SMI handler code which could then be freely modified in theory. An unmodified ThinkPad T60, with similar hardware but retaining the original manufacturer's BIOS, served as control, backup and development system, allowing testing of

 ISBN: 978-1-68558-044-5

SMM code under the Qemu-KVM virtualisation system in conjunction with the related SeaBIOS project[39].

The first performance tests focused on comparing the raw latency penalty imposed by the architecture on transitions between userspace and either kernel mode or SMM as appropriate. This would give an early indication of the viability of the overall approach to explore later, as well as determining how much effort might be required to optimise the design for performance to be viable.

Each test consists of executing the function under test multiple times, recording the elapsed time and calculating the time per iteration from that. To ensure consistency, each test was repeated multiple times and checked for outliers. Timing is measured in two ways: the system 'time of day' clock which records times in microseconds and, for the T60 and virtualised system, the processor Time Stamp Counter read via the 'read time-stamp counter' (`RDTSC`) instruction. On recent Intel processors, including those in use here, the time stamp counter advances at a constant rate regardless of power saving modes or clock speed, making this a useful timing measurement. (On earlier implementations, the TSC rate varied with processor speed, making this usage more problematic.)

The operations tested are listed in Table III. Each set of measurements was performed on each test system, to provide a baseline for interpreting performance figures later (see Section V.C). Table IV shows the test platforms used for benchmarking in the experiments.

TABLE III. OPERATIONS TESTED IN MICRO-BENCHMARKING

Operation	Purpose
NOP SMI	Round trip to/from SMM
open-close	System call requiring access to kernel memory
`getpid()`	Trivial system call to reflect minimal kernel transition cost
signing	Execute a cryptographic operation - specifically generate a signed certificate

TABLE IV. TEST PLATFORMS FOR BENCHMARKING

Model	X200	T60	Qemu-VM
CPU	Core 2 Duo P8400	Core 2 Duo T5600	Core 2 Duo T5600
Clockspeed	2.26 GHz	1.83GHz	1.83GHz
RAM	4 GiB	3 GiB	1 GiB
BIOS	Libreboot	Lenovo original	SeaBIOS

The test code was compiled with level 2 optimisation ('-O2'), for x86-64, in each case. To gather statistical details about the distribution of each individual operation, the test code optionally records the TSC value after each; for the overall operations, to avoid the extra overhead, a consecutive sequence of runs is timed without recording timestamps in between, by compiling with the BATCHONLY flag. For the 1,000,000 iterations of `getpid()`, 8,000,000 bytes of values are written out to memory, almost four times the size of the L2 cache, although writing the values to disk is deferred until after the timed portion. Ordinarily the `getpid()` function is accessed via vDSO for performance reasons— the kernel puts a copy of the PID in the process's own memory space and provides a function to retrieve that directly, avoiding the userspace-kernel round trip, but in order to measure that round trip the legacy system call is used here.

The `getpid()` system call was chosen as the most trivial, since it only copies a non-sensitive constant integer; the open system call will be reading the file system cache, which is not readable from user mode, so incurs greater overhead in a full transition to restore access to kernel data. In normal usage `getpid()` is faster than this, avoiding a system call entirely by returning the process's own copy of this value directly via a mechanism known as Virtual Dynamic Shared Object (vDSO).

The 'signing' test measured a realistic cryptographic operation carried out entirely in SMM. For a web server to be accepted as 'valid' for a given name, it must present a signed certificate asserting ownership of that name, signed by either a trusted root Certificate Authority (CA) directly, or an intermediate certificate which is itself trusted.

This is a two stage process. First, a Certificate Signing Request must be generated, containing a copy of the server's public key and a signature using the private key (the private key itself is never exposed). Secondly, this CSR must be submitted to and accepted by the CA. Originally, this was done manually using human verification of documents and credentials; this still applies for 'Extended Validation' certificates, but for standard 'Domain Validation' certificates this process can now be entirely automatic. Specifically, the free "LetsEncrypt" CA allows ownership of a name to be verified by publishing specific challenge response values in the DNS entries of the name in question, without the server ever having to be publicly accessible. This is one variant of the Automated Certificate Management Environment (ACME) protocol; other variants use the TLS SNI handshake process and HTTP messages respectively to accomplish similar results via other protocols.

This allows a public-private keypair to be generated within the SMM enclave, issued with a valid certificate, then used to host a secured website for testing and demonstration purposes, without ever exposing the key material externally. For testing purposes, however, this external signing step is not necessary: a 'self-signed' certificate is sufficient.

2) Experiment 4 - Webserving

The proof-of-concept webserver application was operated (on the local machine to nullify effects of other network traffic) with three different levels of key isolation: none (a control), process separation, and fully SMM isolated key

handling. In each case the multiple HTTPS requests for pages of differing sizes where pages were automatically generated (via `curl` etc.) and the rate at which requests were served was measured. This allowed a comparison of the relative speeds of the three levels, which are discussed in Section V.D.

V. RESULTS

The results of the four experiments were thus:

A. *Experiment 1 - Basic functionality*

Testing with a range of browsers revealed no significant errors.

B. *Experiment 2 - Protocol compliance verification*

The results of testing with the comprehensive industry standard test suite SSL Labs from Qualys are shown in Figure 3.

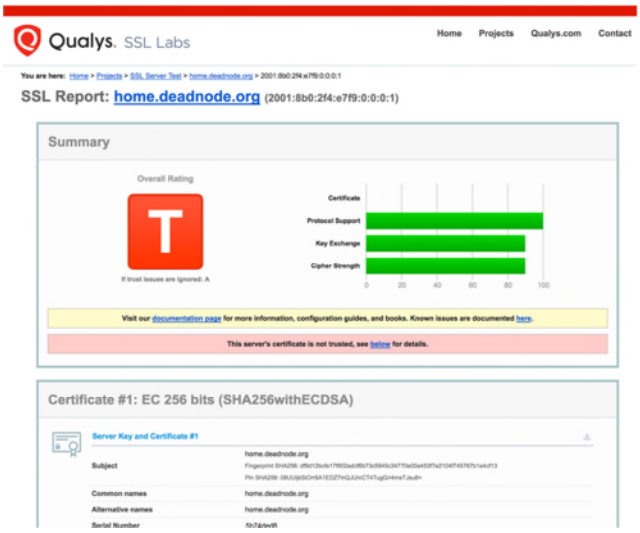

Figure 3. Qualys test suite results

The "T" score indicates a Trust issue — the test server is not configured with a publicly trusted certificate, issued by a genuine Certification Authority such as Verisign or LetsEncrypt — but all cryptographic and protocol aspects are correct; the test suite proceeds to simulate the cryptographic handshakes of a variety of common browsers. With the exception of Google Chrome on Windows XP Service Pack 3, which experiences a handshake failure, all compatible clients negotiate and connect correctly. It is worth noting that no security checks are performed for known vulnerabilities, e.g., Heartbleed etc. – this is purely for compliance with the standard.

C. *Experiment 3 - Microbenchmarking the mode transition cost*

The timing figures obtained are shown in Tables V, VI and VII below. Unfortunately, the X200 system failed during testing, so further results could not be recorded; the remaining tests had to be performed on the fallback system alone, the T60. SMI calls caused the unmodified T60 control laptop to freeze; this appears to be a known, long-standing issue with the stock Lenovo BIOS[40].

TABLE V. EXECUTION TIME FOR SYSTEM CALLS AND SMI INVOCATIONS

Operation	**X200**	**T60**		**T60 Qemu-KVM**	
Units	µs	µs	TSC	µs	TSC
NOP SMI	448	Not available		1310	2.4m
`getpid`	0.4	1.1	620	21	12k
open/close	3	7.1	3900	26	26k
signing	Not available	878	1.606m	905	1.65m

TABLE VI. EXECUTION TIME (TSC TICKS) ON BARE METAL

Operation	**Minimum**	**1st Quartile**	**Median**	**3rd Quartile**	**Maximum**
`getpid`	1133	1155	1155	1155	5211503
open-close	6347	6479	6512	6545	3776872
signing	1534995	1542285.25	1544378	1547757.75	2924856

TABLE VII. EXECUTION TIME (TSC TICKS) UNDER KVM

Operation	**Minimum**	**1st Quartile**	**Median**	**3rd Quartile**	**Maximum**
NOP SMI	2235276	2326436.75	2921712.5	3618389	26339800
`getpid`	20229	20295	20317	20361	33031357
open-close	44902	45397	45496	45595	29565196
signing	1536480	1543069	1546578	1596921	12533972

The relative performance of the two hardware test platforms is indicated by comparing the first two columns indicating the T60 has just under half the speed of the X200 on system calls, while comparing the two pairs of T60 figures ('T60' represents the test code running directly under Linux, 'T60 Qemu-VM' represents the same code executed under Qemu-VM simulation) indicates the relative performance penalty of the simulation system itself: approximately three orders of magnitude slowdown (a factor of 1,000). On the most trivial system call, the additional overhead of simulation dominates (as shown by the much smaller difference between getpid and open/close times), but the relative performance of SMI invocation and open/close calls is more similar: 88 times slower in simulation versus 149 times slower on bare metal.

The maximum times for all operations are extreme outliers — around 3-5 million ticks on bare metal, around four times as high under KVM. Each indicates the test application was interrupted during that operation for between 2-20 ms. The additional KVM overhead is most apparent when comparing the getpid operations (a median more than 17 times slower), closing to a factor of 7 for open-close and no discernable difference on cryptographic operations performed in userspace.

The SMI transition overhead is less uniform, with the upper quartile more than 55% higher than the lower — an interesting characteristic, worthy of further study elsewhere.

 ISBN: 978-1-68558-044-5

One important comparison is between the two full mode transitions (userland/SMM and userland/kernel mode). Since the secured server developed here achieves the security benefits by transitioning into SMM before performing each signing operation, the relative performance impact of this change is indicated by the relationship between the 'signing' and 'SMM' figures: the signature operation in isolation takes a little less than the round-trip to and from SMM, 1.6 million processor ticks versus 2.4 million.

D. Experiment 4 - Performance comparison

The rate of request processing, i.e., the number of requests per second served by the webserver, were measured in three configurations (for a range of response sizes 1KiB-MiB) to identify the additional overhead contributed by the use of SMM to isolate the cryptographic private key and associated code. The control configuration (no isolation at all - so no change of mode - labelled Q0) was compared with the simple option (using a separate user-space process for isolation userland to kernel mode transition - Q1) and the SMM configuration (userland - SMM transition- Q2). The measured rates are shown in Figure 4.

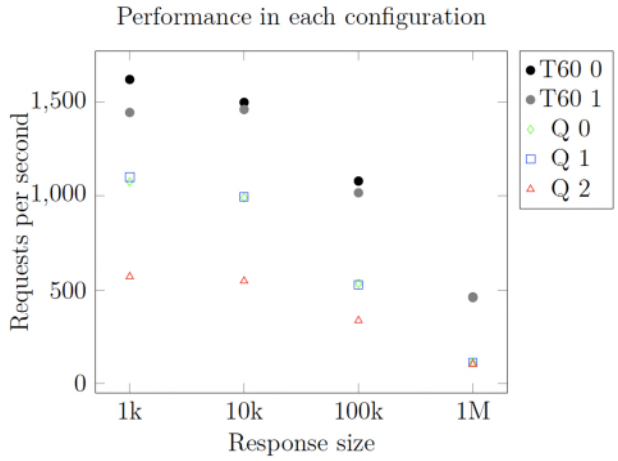

Figure 4. Relative rate of web requests served against response (page) size for each configuration of hardware/enclave type

The performance overhead of simulation as opposed to direct execution is apparent. Across the range of request sizes tested, physical hardware is consistently and proportionally faster than simulated. As the request size increases, the difference between SMM and other modes diminishes to less than 10% at the largest size, one MiB.

VI. Conclusion and Future Work

This work proves the hypothesis: "Secure isolation can be practically implemented using only the long-established Systems Management Mode mechanisms, giving better security isolation than existing techniques such as process separation". In comparison to the baseline approaches (typified by those discussed in Section II) the SMM approach to key-protection has been shown to address their shortcomings and to be robust in circumstances in which they are not. The performance impact of SMM has been explored both on bare hardware and in virtualised form, and a proof-of-concept server demonstrated and benchmarked successfully. Even on relatively old legacy hardware, with additional overhead, the performance impact due to SMM isolation was not prohibitive — approximately doubling the CPU time per handshake operation, causing a performance penalty falling from 50% on the smallest payload sizes (where the handshaking process dominates the overall workload) to 10% at 1 MiB.

A. Implications of results

With a working HTTPS implementation using SMM security, Experiment 4 gave the best indication of SMM's performance impact in the worst case. The relative performance on simulated hardware corroborates the microbenchmark results: performing the cryptographic handshake computations in SMM approximately halves the rate at which handshakes are performed, causing a corresponding slowdown on the smallest requests (where this aspect dominates the overall server performance), falling to around 10% with 1 MiB requests. The effect of size is to be expected: SSL/TLS uses two levels of encryption. First, the connection is established using public key cryptography. This handshake process negotiates two pairs of keys which are then used to encrypt subsequently exchanged data and has a fixed computational cost regardless of the volume of data transferred later. Secondly, the request and response are encrypted using those keys, taking time proportional to the volume involved. So, on small requests the former aspect dominates performance; on larger requests, the latter becomes dominant. The performance shown on the smallest requests, 572 1k requests per second, is also consistent with the bare metal SMM transition measurements from experiment 2 of 448 µs on a processor with approximately twice the performance (a higher clock speed and faster memory bus).

Our results demonstrate the upper bound on the performance or latency cost of isolating the keys in two different ways, validating the original hypothesis about SMM's suitability for this technique. At the smallest extreme of payload sizes, where the cryptographic handshake for each new connection dominates, the additional SMM overhead is of a similar magnitude; as the size increases, the impact of this extra overhead on overall throughput rapidly diminishes.

B. Future work

This work confirms the potential for new uses of SMM in a security context. Unlike reactive patching, SMM isolation provides proactive protection against issues of low-level hardware bugs and protection. Alternative areas for the application of SMM to improve security are discussed below.

1) Intrusion countermeasures

The HyperGuard/HyperCheck projects leveraged SMM as an integrity checking mechanism to detect and alert compromises of a system. These could be incorporated within the application of the SMM: not only would the keys in SMM remain protected, but the compromise would also

be detected and appropriate defensive responses could be triggered.

2) Operation batching

When adopting the SMM approach, significant gains in throughput are expected (in a server situation) from performing multiple cryptographic operations per transition to/- from SMM: rather than passing individual requests immediately, combine the requests into sets and process a full set each time. This would amortise the transition cost across however many connection handshakes are being performed in that batch, trading increased throughput for increased latency determined by the batch size.

3) Other applications and protocols

Particularly with the inclusion of other algorithms, the key protection and handling techniques demonstrated here could be applied to other protocols and applications such as SSH authentication, cryptocurrency transactions or a credential store akin to Microsoft's Credential Guard (which uses a special-purpose virtual machine to isolate credentials from the primary OS on desktop systems).

4) Handshaking overhead in TLS 1.3

The latest version of TLS has a faster handshake than TLS 1.2 used in the experiments but the effect of this on the overhead should be verified.

REFERENCES

[1] Y Kim et al., "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: ACM SIGARCH Computer Architecture News. Vol. 42 3. IEEE Press, pp. 361–372, 2014

[2] M.Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges". In: Black Hat, pp. 7–9, 2015

[3] D. Liu et al., "Architectural support for copy and tamper resistant software". In: ACM SIGPLAN Notices 35.11, pp. 168–177, 2000

[4] G. Chen, et al., "SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution". In: CoRR abs/1802.09085. arXiv: 1802.09085. Url: http://arxiv.org/abs/1802.09085 Retrieved: 2023.06.01.

[5] NVD Spectre – "NVD-CVE-2017-5753 – Spectre", url: https://www.cve.org/CVERecord?id=CVE-2017-5753 Retrieved: 2023.06.0, 2017

[6] J. Sutherland, "On Improving Cybersecurity Through Memory Isolation Using Systems Management Mode", PhD Thesis, Abertay University, Dundee, UK, 2018

[7] F. J. Corbató, M. Merwin-Daggett and R. C. Daley, "An experimental time-sharing system". In: Proceedings of the May 1-3, 1962, spring joint computer conference. ACM, pp. 335–34, 1962

[8] P. J. Denning, "Virtual Memory". In: ACM Comput. Surv. 2.3, pp. 153–189. issn: 0360-0300. doi: 10 . 1145 / 356571 . 356573. url: http://doi.acm.org/10.1145/356571.356573, 1970, Retrieved: 2023.06.01

[9] NVD Heartbleed (2023) - "NVD-CVE-2014-0160 - Heartbleed", url: https://nvd.nist.gov/vuln/detail/CVE-2014-0160 Retrieved: 2023.06.01, 2014

[10] T. Müller, F. C. Freiling and A. Dewald, "TRESOR Runs Encryption Securely Outside RAM." In: USENIX Security Symposium, pp. 17–17, 2011

[11] E-O. Blass and W.Robertson, "TRESOR-HUNT: attacking CPU-bound encryption". In: Proceedings of the 28th Annual Computer Security Applications Conference. ACM, pp. 71–78, 2012

[12] T. Müller, B. Taubmann and F. C. Freiling, "TreVisor". In: Applied Cryptography and Network Security. Springer, pp. 66–83, 2012

[13] T. Shinagawa, et al., "Bitvisor: a thin hypervisor for enforcing i/o device security". In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, pp. 121–130, 2009

[14] J. Götzfried and T. Müller, "ARMORED: CPU-bound Encryption for Android-driven ARM Devices". In: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on. IEEE, pp. 161–168, 2013

[15] T. Müller and M. Spreitzenbarth, "Frost". In: Applied Cryptography and Network Security. Springer, pp. 373–388, 2013

[16] B. Spengler, "PaX: The Guaranteed End of Arbitrary Code Execution" (PDF). grsecurity.net. Slides 22 through 35. Retrieved: 2023.06.01, 2003

[17] N. Provos, "Encrypting Virtual Memory." In: USENIX Security Symposium,pp. 35–44, 2000

[18] N. Sullivan, "Keyless SSL: The Nitty Gritty Technical Details", url: https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/ Retrieved: 2023.06.01, 2014

[19] J.Graham-Cumming, "Incident report on memory leak caused by Cloudflare parser bug", url: https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/ Retrieved: 2023.06.0, 2017

[20] Wikipedia – "Credential Guard" url: https://en.wikipedia.org/wiki/Credential_Guard, Retrieved: 2023.06.0, 2023

[21] S. Bajikar, "Trusted Platform Module (TPM) based Security on Notebook PCs — White Paper". In: Mobile Platforms Group Intel Corporation 1, p. 20., 2002

[22] J. Seo et al., "SGX-Shield: Enabling address space layout randomization for SGX programs", In: Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, 2017

[23] M.Schwarz, S. Weiser, D. Gruss, C. Maurice and S. Mangard, "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: arXiv preprint arXiv:1702.08719, 2017

[24] P-L. Aublin et al., "TaLoS: Secure and transparent TLS termination inside SGX enclaves". In: Imperial College London, Tech. Rep 5, 2017

[25] R. W. and J. Rutkowska, "Attacking SMM memory via Intel CPU cache poisoning". Online: Invisible Things Lab, url: http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf Retrieved: 2023.06.01, 2009

[26] S. Embleton, S. Sparks and C. C. Zou, "SMM rootkit: a new breed of OS independent malware". In: Security and Communication Networks 6.12, pp. 1590–1605, 2013

[27] J. Rutkowska and R. Wojtczuk, "Preventing and detecting Xen hypervisor subversions". In: Blackhat Briefings USA, 2008

[28] J. Wang, A. Stavrou and A. Ghosh, "HyperCheck: A hardware assisted integrity monitor". In: Recent Advances in Intrusion Detection. Springer, pp. 158–177, 2010

[29] B.Ding, Y. He, Y. Wu and Y. Lin, "HyperVerify: a VM-assisted architecture for monitoring hypervisor non-control data". In: Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on. IEEE, pp. 26–34, 2013

[30] K. C. Barde, "Hypervisor security using SMM". US Patent 8,843,742, 2014

[31] A. M. Azab et al., "Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world". In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 90–102, 2014

[32] A. van de Ven et al., "Supervisor mode execution protection", US Patent 9,323,533, 2016

[33] OpenSSL, “Simple TLS Server”, url: https://wiki.openssl.org/index.php/Simple_TLS_Server, Retrieved 2023.06.01, 2022

[34] Google, “Google’s SSL implementation: BoringSSL”, url: https://boringssl.googlesource.com/boringssl/ Retrieved: 2023.06.0, 2022

[35] Qualys, “SSL Labs SSL server test”, url: https : / / www.ssllabs.com/ Retrieved: 2023.06.0, 2014

[36] G. Trent and M. Sake, “WebSTONE: The first generation in HTTP server benchmarking”, 1995

[37] G. Banga and P.Druschel, “Measuring the capacity of a Webserver under realistic loads”. In: World Wide Web 2.1-2, pp. 69–83, 1999

[38] Eclypsium, “System Management Mode Speculative Execution Attacks”, url: https://eclypsium.com/2018/05/17/system-management-modespeculative-execution-attacks/ Retrieved: 2023.06.01, 2018

[39] SeaBIOS Project. “SeaBIOS”, url: https://www.seabios.org/SeaBIOS Retrieved: 2023.06.01.

[40] Ubuntu 2011 - “Lenovo W520 laptop freezes on ACPI-related actions.” url: https : / / bugs.launchpad.net/ubuntu/+source/linux/+bug/776999 Retrieved: 2023.06.01

 ISBN: 978-1-68558-044-5