SAND2019-2489C

Get Your Head Out of the Clouds: The Illusion of Confidentiality & Privacy

Anonymous Author(s)

ABSTRACT

The cloud has been leveraged for many applications across different industries. Despite its popularity, the cloud technologies are still immature. The security implications of cloud computing also dominate the research space. Many confidentiality- and integrity-based (C-I) security controls concerning data-at-rest and data-in-transit are focused on encryption. In the world where social-media platforms transparently gather data about user behaviors and user interests , the need for user privacy and data protection is of the utmost importance. However, how can a user know that his data is safe, that her data is secure, that his data's integrity is upheld; to be confident that her communications only reach the intended recipients? We propose: they can't. Many threats have been hypothesized in the shared-service arena, with many solutions formulated to avert those threats; however, we illustrate that many technologies and standards supporting C-I controls may be ineffective, not just against the adversarial actors, but also against trusted entities. Service providers and malicious insiders can intercept and decrypt network- and host-based data without any guest or user knowledge.

CCS CONCEPTS

• **Security and privacy** → Cloud security;

KEYWORDS

cloud; cybersecurity; privacy; virtual machine introspection; decryption

1 INTRODUCTION

Numerous government and industry entities are initiating the move to cloud [6][20]. In these moves, customers must adopt a shared-security responsibility model [29] with Cloud Service Providers (CSP). The shared security responsibility model is an agreement between the customer and the CSP that each has security responsibilities that they must independently perform, typically with no overlap in security functions. Thus, the customer must place significant trust in the CSP and its ability to protect their data. The CSP is expected to implement appropriate measures designed to secure customer content against accidental or unlawful loss, access, or disclosure in its cloud infrastructure, implementing security measures for system layers and applications for which it is responsible. Likewise, the customer is responsible for implementing appropriate measures to protect its data for the system layers and functions for which it is responsible. The separation in the system layers for which the CSP is responsible versus where the customer is responsible is distinguished by the type of cloud service provided.

For services such as Software as A Service (SaaS) or Platform as a Service (PaaS), the CSP will have either access to unencrypted data or the keys that are used to encrypt the data [15]. In the case of data storage, the CSP may require an encrypted link between the customer and the cloud; for this protection, the CSP has knowledge of the key or certificate used to encrypt the data. For example, when data is stored in Amazon Web Service (AWS) S3 buckets, the customer may encrypt the data using customer-controlled encryption keys. However, AWS requires the customer to share the encryption key with them, which is in turn used to generate a hash-based authentication code (HMAC) for verification purposes [3]. In this example, AWS states they destroy the encryption key material [3] after the data has been stored (when using customer server-side encryption). Thus, the customer must place significant trust in the CSP, both in the CSP honoring their claims and the effectiveness of CSP security processes. The customer acknowledges this trust model with the CSP and accepts the additional risk.

According to the National Institute of Standards and Tecnology (NIST), security controls for confidentially and integrity regarding data-at-rest and data-in-transit are based on encryption [28]. The principle being, to protect the data (its confidently and integrity), encrypt the data and ensure only those who are authorized can access this information. System layers in cloud are afforded encryption and authentication mechanisms controlled and managed by the CSP, particularly for SaaS and PaaS. However, many CSPs also have Infrastructure as A Service (IaaS) offerings which provide opportunity for the customer to have full control of data encryption mechanisms. In IaaS, the customer can choose to generate their own key pairs using industry-standard tools like OpenSSL. Here, the customer generates the key pair in their own secure and trusted environment. The customer then provides only the public portion of the key pair to an IaaS instance located in the cloud. Data can then be moved securely between the customer's trusted environment and the IaaS instance running customer-configured security software. In the case of IaaS, the plaintext data residing on the cloud instance is securely located in cloud virtual machine processes. The CSP should have no method to access the data in the customer-configured and -instrumented IaaS instance. But does the CSP have access to the guest operating system? Can the CSP access either plaintext data in the guest operating system (OS), or the encryption keys in the guest OS? If the CSP has undetectable access to the IaaS instance, then the cloud customer cannot expect privacy even in the case of IaaS.

The authors of this research paper have developed Virtual Machine Inspection (VMI) capabilities for advanced computer security purposes [2]. These same VMI capabilities could be used by a CSP to instrument its IaaS instances. If the VMI capability is used in the CSP cloud offering, the customer should recognize that such technology may enable a CSP or malicious insider access the plaintext of their encrypted data. Furthermore, CSP access to the data may be completely undetectable by the cloud customer using IaaS services.

The contributions of this paper to the community are as follows:

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

- The authors will describe how network and disk encryption in the cloud may not be sufficient to protect the confidentiality and integrity of user data
- The authors will describe a hypervisor agnostic mechanism to transparently access information, in a real-time fashion without the tenant's knowledge and with negligible performance impact to the tenant
- The authors will describe several mitigations that should be sought after to increase privacy

The remainder of this paper starts with a technical description of the VMI technology and its uses. Next, a section describing specific applications that are commonly used in CSP IaaS deployments and how VMI can be used by the CSP to eavesdrop on encrypted data. This section discusses key extractions and undetectable deployments by the CSP. This is followed by an experimental analysis executed to show, with statistical strength, that eavesdropping through VMI is practically undetectable from the guest. Finally, the authors conclude with suggestions for mitigating the risk of unwarranted decryption of customer data.

2 BACKGROUND

The National Institute for Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [30].

Several CSPs have emerged as industry leaders as cloud computing has become increasingly mature and available. Amazon, Google and Microsoft have demonstrated support through the promotion, encouragement, adoption, and leadership of cloud computing, building a foundation for recent paradigm shifts. The paradigm will continue to evolve as the cloud becomes more pervasive.

The promise of cloud computing has spurred entrepreneurial development of cloud services. The services provided by these businesses are generally divided into three categories:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

With both SaaS and PaaS, CSPs often have tight control of the execution environments, as the applications that users access are limited in the number of configurable options. The focus of this paper is on the infrastructure component, IaaS. There are several IaaS platforms (e.g., OpenStack, OpenShift, AWS, Microsoft Azure) as well as providers in both private and public settings. IaaS provides users with the most freedom of configuration for their virtual environments, comparable to what they would have in their own enterprises. However, just as in traditional networks, IaaS is not immune to malicious actors that take advantage of poor security policies, weak credentialing, and multi-tenancy, let alone the malicious insider who walks by equipment racks daily. Still, companies and government sectors have made it part of their long-term strategic plans to employ cloud technologies for their infrastructures [16].

In this sense, cloud has been leveraged for many applications by many different industries. Despite its popularity, the development and impact of cloud technologies are still immature and are thus open areas for research and development [24][1]. The security implications of cloud computing are a rich topic that also dominates the research space. From a forensic perspective, numerous questions arise on how to analyze cloud resources using traditional digital forensics techniques [37][38]. For instance, during a traditional digital forensic examination, all files on the storage media are examined along with the entire filesystem structure. However, this is not a practical model for cloud infrastructure, as the elasticity and ephemerality of pooled storage make pinpointing data blocks cumbersome. This difficulty is exacerbated in networked systems by the scale with which computing resources are spread over diverse administrative and geopolitical domains. Cloud computing can combine numerous heterogeneous resources (hardware platforms, storage back ends, filesystems) that may be geographically distributed. The idiosyncrasies in cloud environments have caused a paradigm shift in digital forensics; however, tools and techniques still do not exist to help forensic practitioners cope with these issues. And while many research areas enumerate these challenges, open literature has not made significant headway to address the issues or provide solutions.

The set of security and privacy objectives of an organization, therefore, is a key factor for decisions about outsourcing information technology services, and for decisions about transitioning organizational resources to a public cloud and a specific provider's services and service arrangements. What works for one organization may not necessarily work for another. In addition, practical considerations apply - most organizations cannot afford financially to protect all computational resources and assets at the highest degree possible and must prioritize available options based on cost as well as criticality and sensitivity. When considering the potential benefits of public cloud computing, it is important to keep the organizational security and privacy objectives in mind and to act accordingly. Ultimately, a decision on cloud computing rests on a risk analysis of the trade-offs involved. Much of the risk analysis on data security and privacy centers around the notion of encryption. The NIST states that "Data must be secured while at rest, in transit, and in use, and access to the data must be controlled. Standards for communications protocols and public key certificates allow data transfers to be protected using cryptography." [16]

The adoption of cloud varies significantly between organizations for numerous reason ranging from the purpose, legal obligations, exposure to the public, threats faced, and tolerance to risk. From a risk perspective, determining what cloud services an organization can adopt is not possible without understanding the sensitively of the data that is being moved and the threats that the organization is facing. Further complicating this issue is that the risk and threats are becoming more prevalent and aggressive. In the last year, attacks such as Spectre and Meltdown have elucidated key issues in how hardware-based attacks can significantly reduce the trust in co-used hardware (such those used in IaaS platforms) [31]. Meltdown and Spectre affects personal computers, mobile devices, and cloud; depending on the CSP's infrastructure, it may even be possible to steal data from other customers. Three years ago, many

thought that information once stored in RAM, would stay in RAM was physically defective. Or (if one were a DRAM manufacturer), that it could change sometimes, but this was simply a reliability annoyance. The Rowhammer work [12][27] demonstrated that DRAM disturbances could allow root access on a fully updated OS without exploiting any software vulnerabilities, i.e., just flipping enough bits to get write access to users page tables. That "couldn't" happen - until it did. In the face of Spectre and Meltdown, most believed that an unprivileged user-land process could not read out all of system physical memory without any indication as to what it was doing. Today? Not so much. The realization is that operating systems rely on hardware keeping the guarantees the architecture promises, such as memory isolation (that one process cannot read the memory of another process). When the hardware doesn't keep those guarantees, nothing else can build a secure platform on top of it. Despite all sorts of reasoning about process isolation and virtual memory, over the past decade change has brought *global read access* to everything if asked for properly. However, even with these attacks, the mindset is that we still trust CSPs to protect our private data.

Cloud computing is an outgrowth of the advances in virtualization technologies. Through virtualization, a layer of opacity is injected between the perspective of the user on the guest, and the perspective from the hypervisor - often called the symantic gap. VMI development is a fundamental technology associated with computer virtualization, meant to bridge that symantic gap. Thus, the applicability of VMI to cloud computing services is a straightforward application. The remainder of the paper continues with descriptions of key aspects of VMI and how they tie into the cloud computing paradigm, and how they can reduce or eliminate cloud user privacy. As cloud computing, and specifically IaaS, become more ubiquitous, it becomes more imperative to address the risks that follow.

3 THWARTING ENCRYPTION AT-REST AND IN-MOTION

Privacy and confidentiality in the cloud environment are based on the user's trust of the CSP. With respect to IaaS, the user may have more confidence with their trust since they own a greater portion of the confidentiality piece: encryption. However, the authors argue that through introspection from the host machine, that trust may be all for not.

3.1 Virtual Machine Introspection

Virtual Machine Introspection, a term coined by [11] in 2003, was originally meant to provide an architecture to support an intrusion detection systems (IDS) based on the premise that a network-based IDS would be more resistant to attack but would have a diminished view of what would be happening on the host. The authors referred to it as a Virtual Machine Monitor (VMM) and built their first prototype named Livewire. Nowadays the term has become more general, wherein VMI is used to monitor low-level details of a running virtual machine by viewing its memory, trapping on hardware events, and accessing the vCPU registers [33].

The application of VMI has been used to augment network-based security systems, enforce security policies on VMs, monitor VM performance, and study/identify cybersecurity matters such as malware execution. Applications based on VMI methodology include LibVMI [33], and its variants that include Volatility [9] or Rekall [36] for advanced memory forensics. Various commercial hypervisors have also seen the incorporation of VMI capabilities throughout the years [34][42][7][41][35], toward the end of improving cybersecurity and visibility for the events occurring on virtual machines.

Other efforts in the research domain employ VMI for specific use-cases, such as application whitelisting [14], page-level binary code signing [22], heap allocation monitoring [17][21], and interactive system call redirection [10][45]. The authors of [44] have developed CloudVMI, which provides inspection and manipulation of the state of VMs, particularly in a cloud environment. Their solution attempts to provide VMI as a service for cloud users but is based heavily on Xen and LibVMI. The authors of [43] also target cloud environments with their VMI product. Their solution attempts to overcome intrusiveness, latency and OS-dependence found in other VMI applications. Their claim is a real-time, transparent VMI solution that is event-driven and generates snapshots cooperatively to reconstruct guest memory states. However, their most recent prototype is tooled for the KVM/QEMU hypervisor and targets only 32-bit Linux-based operating systems. The authors of [39][40] have used VMI for extracting encryption keys from the VMs to aid IDS systems and identify malicious network traffic. Their approach is based on walking memory structures associated with the guest VM to locate encryption keys (such as RSA) through a Forensic Virtual-Machine Framework (FVM). They describe a very detailed, in-depth methodology to locate keys in VM memory snapshots; however, their evaluation relied on a single OS and hypervisor, subject to storage constraints. The authors of [4] describe their product Goalkeeper as a VMI tool used to enforce cybersecurity policy through application whitelisting, wherein the VMI tool terminates policy violators using a customizable set of VMI-based process termination techniques. Their VMI tool was developed for the Xen hypervisor, but only supports Linux OS (due to source code and debugging symbol availability).

The authors of this paper have also developed a VMI capability. The novelty of their VMI tool is that is deployable against various hypervisors and guest operating systems, with the same attributes of transparency to the guest user. Rather than integrate with the code base of a hypervisor, and/or rely on open-source code/symbols for VM operating systems - the VMI tool is implemented as a shim process that augments unmodified hypervisor code. Further, on the topic of kernel symbols and operating system idiosyncrasies, it dynamically identifies attributes (symbols) of various operating systems on-the-fly through analyzing memory structures upon first encountering the VM. So, while many of the aims of VMI in research have been to aid the cybersecurity practitioner, it is authors' observation that it may be the case that security is not what VMI need be ultimately used for. The flexibility of VMI, and its goal of being transparent to guest processes, should sound the alarm for clients of CSPs.

3.2 Understanding VMI

There are many different methodologies in place to perform VMI. The authors of [13] describe four categories for VMI:

(1) In-VM: monitors the guest OS from inside the VM and exposes its activities to the hypervisor (who in turn enables enforcement of policy).
(2) Out-of-VM Delivered: passive VMI, uses "delivered" semantic information from the guest via explicitly incorporating data in the VMI system, extracted from OS source code, or obtained through kernel symbols.
(3) Out-of-VM Derived: uses hardware provided functions to observer/gather guest events and states, by either handling traps (caused natively by the guest OS), or by forcing traps using hardware-based hooks.
(4) Hybrid Techniques: use a combination of any of the three methods above.

The VMI approach described herein is largely based on the third category, Out-of-VM Derived. By inserting a hook anywhere within the handling of a VM-Exit by a hypervisor, the VMI tool can gain execution in Virtual Machine eXtentions (VMX) mode, and thus control execution state and read memory of the currently exited VM. These two primitives are enough to gain access to any data processed by that VM. Some actions only require memory read abilities, while others require an ability to modify state to cause VM-Exits in circumstances where they would not have otherwise happened.

Some hypervisors are not protected from malicious code insertion into their execution path, nor from the preemption of their VM-Exit handler. Even those that are segmented from the rest of the system offer APIs that allow for some access to the hypervisor's capabilities to less-privileged code.

Simply having the ability to read guest memory provides some capabilities to extract information from the guest. To read key material from a Firefox session, it is necessary to parse PE files, identify processes, and modify execution flow by changing the `GUEST_RIP` field of the Virtual Machine Control Structure (VMCS). However, to obtain keying material from a security policy system (such as Window's Local Security Authority Subsystem Service (LSASS)), a naïve memory-reading algorithm that has full access to host memory, but knows nothing about virtualization, is sufficient. It would likely produce false positives, but since each session key also comes with an identifier to match the correct stream, this turns out to be a non-issue. The overhead of scanning all guest memory would also be much higher, and since there are no cases yet where the VMI tool does not have access to data from the hypervisor context, the authors chose the lightweight route for an LSASS proof of concept.

3.3 By-passing Encryption through Hypervisor-assisted VMI

By creating some basic VMI and debug-like capabilities, any hypervisor can be used to bypass encryption. This is a fundamental ability since the hypervisor has direct visibility and control over guest execution. For example, any VMI capabilities that include the ability to set breakpoints or log system calls is easily used to bypass typical file encryption by simply gathering input/output (IO) buffers before they are passed from user space to the kernel (before hitting the encryption layer).

The problem extends far beyond simple dumping of unencrypted buffers. To better illustrate this, the authors' VMI implemented a handful of basic abilities in several hypervisors for both Windows and Linux guest introspection. An API was created to set breakpoints, parse binaries, enumerate process information, and hook any or all system calls. This was done without utilizing any APIs provided by the hypervisors (as the provided APIs were limited both in speed and capability). By implementing custom interfaces and introspection code that attached to the hypervisors, the VMI tool gains capabilities that were otherwise not offered by the hypervisor authors. This set of capabilities allows the VMI user to do interesting tasks (like those of passive introspection), but also provides the ability to change guest state or execution - including full control over hardware-defined control structures like the VMCS.

Given these VMI capabilities, several examples of monitoring a guest VM to gather data that would otherwise be encrypted or unavailable outside the VM are presented.

First, extracting SSL session keys in real-time from a web browser. In this example, the VMI identifies a web browser process starting, then enumerates its loaded DLLs, finds addresses of imported/exported functions, and sets extended page table (EPT)-based breakpoints in several strategic places. The VMI application also changes execution by modifying the VMCS and redirecting the guest instruction pointer to a function which would not otherwise be used.

Next, the authors discuss how through memory scanning alone, the VMI tool can enumerate all session keys used by client and server applications utilizing Microsoft's Cryptography API: Next Generation (CNG), and how this may be improved by VMI.

Finally, the discussion delves into the ability to intercept system calls, through which the VMI tool can gather data being read from or written to encrypted volumes such as VeraCrypt (and others) on both Windows and Linux.

It is also worth noting that these are done in a hardware virtualized environment in real-time, rather than with emulation.

3.4 Extracting TLS Session Keys from a Browser

There are likely many ways to accomplish this task; the authors present a method that requires little understanding of the Firefox code, and without the need to re-implement any cryptography code. This is meant to illustrate the ease with which a hypervisor can extract data from a running system without its cooperation or knowledge. The hypervisor can be made to cause the guest VM to execute code that it would not otherwise execute and do so without introducing any easily noticeable artifacts.

Firefox contains an optional feature that allows SSL session keys to be extracted in real-time. This feature is not enabled by default and may even be compiled out. A quick look through the Firefox source code shows a function named `PK11_ExtractKeyValue` is used by this feature. It takes a single argument which is a pointer to a `PK11SymKey` structure. This function decodes a key value which is later dumped to the log file. As it turns out, this code path is not enabled by default, and is never run unless the `SSLKEYLOGFILE` environment variable is set to a valid value upon Firefox startup.

Next, the identification of keys of interest is carried out so they may be passed to `PK11_ExtractKeyValue`. Two more functions are also leveraged: `PK11_PubWrapSymKey` and `PK11_FreeSymKey`. Setting a breakpoint on `PK11_PubWrapSymKey` allows the VMI tool to numerate addresses of any `PK11SymKey` structures that represent a symmetric key. Finally, by also placing a breakpoint on `PK11_FreeSymKey` and looking for references to known symmetric keys, the VMI user may be notified when Firefox is finished using those keys.

After setting these two breakpoints in a callback for the `PK11_FreeSymKey` breakpoint, an emulated a call to `PK11_ExtractKeyValue` is executed by pushing the current instruction pointer value onto the guest VM stack and writing the address of `PK11_ExtractKeyValue` to the `GUEST_RIP` field of the VMCS. This, in effect, performs a call to `PK11_ExtractKeyValue` on the `PK11SymKey` that were just about to be freed, and then returns to `PK11_FreeSymKey` to immediately free the key, where a break is affected againâĂŤbut this time, the VMI tool gathers the decoded value from the structure before letting it continue into the `PK11_FreeSymKey` function. After some simple formatting, the data is identical to what would appear in the NSS log, all without enabling the feature in the guest VM [26] . As mentioned, this is all done in real-time on a live system with no notification to the user that this is happening.

3.5 Extracting Keys through LSASS

On a modern Windows platform, most applications that wish to use built-in cryptography functionality will use the Cryptography API: Next Generation (CNG). Some of the most common applications include Edge, Internet Explorer, and Remote Desktop Services. CNG is used by server applications as well.

In [18], techniques are presented for gathering session keys used by client and server processes that utilize CNG. This was originally done using Volatility [9] or Rekall [36] (though to the authors' knowledge the source was never publicly released). The basic abilities needed to gather session keys require almost no bridging of the semantic gap, though doing so drastically improves performance. The authors' adapted the techniques to work from the context of a hypervisor and used knowledge of the EPROCESS structure in Windows to link the name lsass.exe to the directory table base (a.k.a. the page table base or CR3) of the LSASS process (the Local Security Authority Subsystem Service). As described in [18], this process acts as a repository for all session keys. By isolating the guest's page tables and with access to the Extended Page Tables (EPT) from the hypervisor side, the VMI tool can enumerate the ring3-available memory in the LSASS process and walk EPT to determine the host addresses that correspond. This leaves a small subset of the overall guest memory to search through, and by searching for the necessary markers, a poll can be made against this set of memory from outside the guest to recover all session keys stored in LSASS. The process required to locate these keys is to find sets of three linked structures, two of which have a 4-byte magic value, and the other a C++ object which has one of two fixed virtual function tables located at the beginning.

This code is implemented to run in a hypervisor context that periodically scans guest memory for any new copies of the structures that may have been created since the previous run. Since it is system context, the search is limited to memory inside the ring-3 portion of LSASS (where periodic re-enumerate of the range is done). Actions also include attempts to pre-determine the two harder-to-locate virtual function table pointers, since they do not change throughout a single boot of each Windows system. Finally, another thread can poll the current memory set at an arbitrarily chosen interval to search for the objects that contain the session keys. All key material is then dumped to a file on the host which can be parsed by tools like RDP-Replay, Wireshark, or the inline decryption chain described below.

3.6 Passing Keys for Decryption In-Motion

The following approach carries on from the key extraction methods above, and results in the live decryption of traffic generated within an IaaS cloud environment. The decryption of network traffic has often been seen in two contexts: (1) in a malicious manner, where an actor may attempt to employ man-in-the-middle (MITM) attacks to gain information or payload data [32]; and (2) in an enterprise network, where system administrators leverage bump-in-the-wire technology to decrypt network traffic for payload inspection, then re-encrypt upon egress [8]. Both methods involve direct interaction and modification of the network stream to effect decryption and subsequent re-encryption. The approach described here avoids these methods by directly mirroring the encrypted traffic stream, causing no disruption to the flow of traffic. Furthermore, through VMI, the necessary key material to decrypt those streams out-of-band is extracted in real-time. Figure 1 displays the general architecture and workflow used to implement this model.

As an example, a user is logged into their cloud IaaS instance and visits a HTTPS enabled website through their browser, as show in Figure 1. Traffic from that session is ushered through the cloud infrastructure network and then output to the Internet (through a virtual switch, OVS). At step (1) the VMI application identifies the establishment of an encrypted web session, and subsequently begins extraction and storage of the key material (2). Thus, the CSP has obtained possession of the user's private keying material and can then access its network (4) for the encrypted data stream and use the key material from (1) to decrypt the stream (3). The keying material is then applied to the traffic which may be reassembled as an unencrypted HTTP stream. The unencrypted stream is then output and made available to the CSP to be stored as cleartext; simple tools may be also be applied to streams for the reconstruction binaries as well (5).

3.6.1 Implementation Details. The following describes an architecture and process for decrypting traffic on-the-fly that may be used at a hosting entity. The process of decryption begins with a priori knowledge of the VM targeted, and the IP address of the network interface on that VM. Each VM in the environment is identified by a unique PID, as assigned by the hypervisor and host operating system. The VMI outputs specific data streams, written to files on the host machine; the output for keys is instantiated and filtered by the VM PID (`host_pid`) and written to a unique file for storage. Example key entries are shown below:

```
1493913648497720217,host_pid=aea,vpid=0,key=RSA ae673
37de13a127b 03038c66e9c5a43623812e467bf7ad8113058e9a06
```

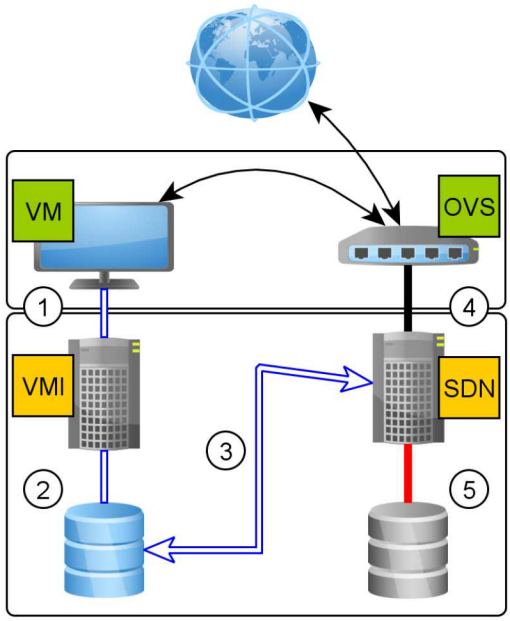

Figure 1: Live Decryption Workflow

```
c1cea8b91a1cf50d62c2e7fa0d5c8955a14a9f4be03173d60d8473

1493913648695743196,host_pid=aea,vpid=0,key=RSA 59293
553b4f9cae3 0303e8d4d386387fc7e290786f8d945d161f4b7df9
b6bdbff77fb7b85d1eb482a71124292ac9ec1788776cf9b4635ab1

1493913648854945716,host_pid=aea,vpid=0,key=RSA 1534e
4d05b402d00 030304448941bbfc02fa8c33534f5d4fd32027707d
275bcef046ec36c80261a9f45e552dbc6b1ca5bf80d7aa57a0f69d

1493913649118681514,host_pid=aea,vpid=0,key=RSA 7aa94
450da746006 030367a8cf1758e3738ea25780fc09681d6cc12c97
e66b1c5f3aeb6b7d11c6d827f5a5e75cc01496a31d1c27a0174a22

1493913649405792039,host_pid=aea,vpid=0,key=RSA a67b9
d42e379d850 0303d77d349114386f8e863ed0cf57fcb9d0bc6cd0
d68568f1521b6934d84fcfe869698a8649608aa18cafdd4e39646f
...
```

The output data consists of the epoch timestamp, the `host_pid` identifier for the VM, the virtual PID (virtual CPU) identifier and, finally, the key data. The key data consists of the crypto standard, an 8-byte value followed by a 48-byte value, the "pre-master secret." The first smaller value is the first 8 bytes of the larger value as it is seen encrypted over the wire. This format is prescribed by NSS [23] to expedite the match to the proper key exchange; that is, matching up the first 8 bytes of the RSA encrypted premaster secret in the client key exchange packet will identify the key that decrypts with the following 48-byte RSA key.

Concurrently, a command is sent to the switching fabric (e.g., Software Defined Network (SDN) controller) to initialize capture of traffic to and from the VM. (Note: CSPs often deploy SDN technologies within the cloud infrastructure to perform flow-based networking). In the CSP's infrastructure, the SDN controller begins by creating two virtual interfaces on an the virtual host switch: `delay_tap` and `decrypt_tap`. A port mirror is established on the VM's virtual interface to the `delay_tap` interface. A traffic control (tc) profile is then applied to the `delay_tap` interface to delay the flow of traffic by an arbitrary duration; this is done to provide adequate time to consume and apply the key material to the stream. The decryption function is then fed the `delay_tap` as the source of traffic, the keying material storage location, and additional flags to further filter the traffic (such as IP, ports, and traffic type, e.g., HTTP and SSL). Raw, unencrypted traffic (2) is then output to the packetizer function to reconstruct the stream.

```
Decrypted SSL data (583 bytes):
0000  47 45 54 20 2f 63 6c 69 65 6e 74 5f 73 74 6f 72   GET /client_stor
0010  61 67 65 2f 61 31 32 35 33 37 35 35 30 39 2e 68   age/a125375509.h
0020  74 6d 6c 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f   tml HTTP/1.1..Ho
0030  73 74 3a 20 61 31 32 35 33 37 35 35 30 39 2e 63   st: a125375509.c
0040  64 6e 2e 6f 70 74 69 6d 69 7a 65 6c 79 2e 63 6f   dn.optimizely.co
0050  6d 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d   m..User-Agent: M
0060  6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 57 69 6e 64   ozilla/5.0 (Wind
0070  6f 77 73 20 4e 54 20 36 2e 31 3b 20 57 69 6e 36   ows NT 6.1; Win6
0080  34 3b 20 78 36 34 3b 20 72 76 3a 35 33 2e 30 29   4; x64; rv:53.0)
0090  20 47 65 63 6b 6f 2f 32 30 31 30 30 31 30 31 20    Gecko/20100101
00a0  46 69 72 65 66 6f 78 2f 35 33 2e 30 0d 0a 41 63   Firefox/53.0..Ac
00b0  63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d 6c 2c   cept: text/html,
00c0  61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 68 74 6d   application/xhtm
00d0  6c 2b 78 6d 6c 2c 61 70 70 6c 69 63 61 74 69 6f   l+xml,applicatio
00e0  6e 2f 78 6d 6c 3b 71 3d 30 2e 39 2c 2a 2f 2a 3b   n/xml;q=0.9,*/*;
00f0  71 3d 30 2e 38 0d 0a 41 63 63 65 70 74 2d 4c 61   q=0.8..Accept-La
0100  6e 67 75 61 67 65 3a 20 65 6e 2d 55 53 2c 65 6e   nguage: en-US,en
0110  3b 71 3d 30 2e 35 0d 0a 41 63 63 65 70 74 2d 45   ;q=0.5..Accept-E
0120  6e 63 6f 64 69 6e 67 3a 20 67 7a 69 70 2c 20 64   ncoding: gzip, d
0130  65 66 6c 61 74 65 2c 20 62 72 0d 0a 52 65 66 65   eflate, br..Refe
0140  72 65 72 3a 20 68 74 74 70 3a 2f 2f 77 77 77 2e   rer: http://www.
0150  63 6e 6e 2e 63 6f 6d 2f 0d 0a 43 6f 6f 6b 69 65   cnn.com/..Cookie
0160  3a 20 63 64 6e 3d 68 74 74 70 25 33 61 25 32 66   : cdn=http%3a%2f
0170  25 32 66 61 6b 61 6d 61 69 25 33 61 64 73 64 25   %2fakamai%3adsd%
0180  34 30 63 64 6e 2e 6f 70 74 69 6d 69 7a 65 6c 79   40cdn.optimizely
0190  2e 63 6f 6d 25 32 66 6a 73 25 32 66 31 33 31 37   .com%2fjs%2f1317
01a0  38 38 30 35 33 2e 6a 73 0d 0a 43 6f 6e 6e 65 63   88053.js..Connec
01b0  74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65   tion: keep-alive
01c0  0d 0a 55 70 67 72 61 64 65 2d 49 6e 73 65 63 75   ..Upgrade-Insecu
01d0  72 65 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a   re-Requests: 1..
01e0  49 66 2d 4d 6f 64 69 66 69 65 64 2d 53 69 6e 63   If-Modified-Sinc
01f0  65 3a 20 54 75 65 2c 20 32 33 20 4d 61 79 20 32   e: Tue, 23 May 2
0200  30 31 37 20 30 30 3a 35 32 3a 32 38 20 47 4d 54   017 00:52:28 GMT
0210  0d 0a 49 66 2d 4e 6f 6e 65 2d 4d 61 74 63 68 3a   ..If-None-Match:
0220  20 22 37 63 38 65 38 62 38 39 37 64 37 39 39 31    "7c8e8b897d7991
0230  34 35 38 65 62 30 38 31 33 39 33 65 35 31 63 33   458eb081393e51c3
0240  31 66 22 0d 0a 0d 0a                              1f"....
```

Figure 2: Raw Unencrypted Traffic

Each block of plaintext network traffic as output from the decryption function are classified as either frame, reassembled TCP, or decrypted SSL. A packetizer function consumes each of these classified blocks to reconstruct entire TCP segments and streams, maintaining all the application layer flag and option settings, whilst updating the lower level attributes (such as CRC calculations). This updated, plaintext session traffic is then output to the decrypt_tap.

The traffic leaving the `decrypt_tap` may be consumed by two functions: (1) a packet capture function, can facilitate full reconstruction of payload data as required, and (2) a deep packet inspection (DPI) function, which can summarize the attributes of the traffic flow and extract relevant payload (layer-7) metadata as required. The latter DPI function can output the data to a SIEM, which provides correlation between other traffic in the network, or to other host data in the cloud environment.

3.7 Transparently Decrypting Filesystems

By simply monitoring `sys_read` on Linux or `NtReadFile` on Windows, the VMI tool can gather data out of file buffers on most types of encrypted filesystems. This is because the encryption generally takes place at a lower layer than that of the system call.

When encrypted files are at rest, their contents are not copied into buffers and passed to system calls (where a VMI application might try read out their contents). However, by modifying arguments or redirecting guest execution, the system may be tricked into reading a file it otherwise would not have touched. Thus, the authors approach allows the VMI to read the contents of arbitrary files on an encrypted volume at will.

3.8 Alternate Approaches

The examples shown above purposely modify execution flow in the guest to illustrate the ease with which it can be used to produce desired outcomes with guest data. However, numerous alternatives exist. Though it would require more effort, it would also be possible to passively gather encoded keys and decode them by writing custom extraction routines. Additionally, the VMI tool could weaken the random number generator or other parts of the session key generation routine, or perhaps find a way to gather session keys when they are shared over network connections. In addition, it has been shown that having full physical memory access to guest VMs is enough alone to identify cryptographic keys in memory [23].

4 PERFORMANCE TESTING: WHAT DOES TRANSPARENT MEAN?

A key questions is, "Can the cloud user detect if VMI is being used by the CSP on a specific IaaS instance?" One hypothesis is that VMI may impact VM or instance performance where the user can recognize the performance impact. An experiment was designed to show that a user could not identify VMI was being used on their VM to extract key material and decryption traffic on-the-fly.

The design of the experiment is one based on factorial analysis. Two experiments were run, each involving access to encrypted data at-rest and in-motion. The experiments were run on hardware often used by CSPs in their cloud offerings. Experiment orchestration was performed with minimega [25].

4.1 Testing Parameters

The host servers used were Supermicro blade-servers running Ubuntu Server 16.04 LTS with 256GB memory and 32-core Intel Xeon E5-2650 processors. The virtualization manager used was minimega [25], leveraging KVM as the hypervisor. The virtual webserver was a nginx process run from a lightweight Linux virtual machine (tiny core) with 16GB memory and four virtual CPUs. The endpoints (clients) were Windows 10 x64 virtual machines running Firefox 64-bit, each with 8GB memory and one virtual CPU.

4.1.1 Network Decryption Experiment. The network decryption experiment consisted of two VMs running in a client-server configuration. The client VM (Windows) web browser was automated to visit the server VM (tiny core) to retrieve various files of different size over a TLS-encrypted channel. The host machine ran: (1) without VMI, (2) with VMI, extracting keys from Firefox (F), (3) with VMI, extracting keys from LSASS (L). For VMI tests, the keys were extracted then used to decrypt network traffic, as described in Section 3.6 above. Trials were run over 30 iterations to meet a normal distribution. The response variable (metric) was time in milliseconds.

The factors and levels considered for the analysis of this experiment can be seen in Table 1.

Table 1: Network Test Factors

Factor-1
{No VMI, FF Key Extraction, LSASS Key Extraction}

Factor-2	**Factor-2 Levels**
Payload size	{1K, 500KB, 1MB}
Cipher suite	{(*),(+)}

*TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
+TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA

4.1.2 Host Decryption Experiment. The host decryption experiment consisted of a single VM (Windows). The experiment exercised the VMI's capability to extract encrypted files written and read from the OS's filesystem. Files were generated from randomized data, then encrypted using Windows EFS. Writes were accomplished by copying the file; reads were accomplished by concatenating the file. Each test was run 30 times to meet a normal distribution. The response variable (metric) was time in milliseconds.

The factors and levels considered for the analysis of this experiment can be seen in Table 2.

Table 2: Host Test Factors

Factor-1
{No VMI, With VMI}

Factor-2	**Factor-2 Levels**
File Read	{1MB, 10MB, 100MB}
File Write	{1MB, 10MB, 100MB}

4.2 Results

The results of the two experiments are discussed in the following sections.

4.2.1 Network Decryption Results. The Network Decryption Experiment consisted of establishing a TLS-encrypted channel between two VMs: a client and server. The TLS-encrypted channel was initialized by the client's web browser and used to download files of three different sizes (1KB, 500KB, and 1MB). Two ciphersuites were used for the TLS connection, as commonly used for TLS v1.2 (*) and TLS v1.1 (+). Introspection factors included: no VMI, VMI using Firefox key extraction method, and VMI using LSASS key extraction method. The times to download the files, based on combination the of factors is graphically represented in Figure 3. Each trial was run 30 times.

In Figure 3, the x-axis represents the number of bytes transferred (file size); the y-axis represents the average time to download. As can be seen, the times for each factor combination are almost identical.

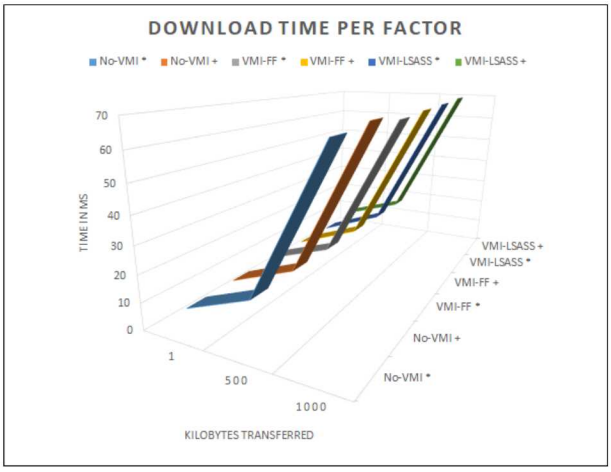

Figure 3: Download Time per Factor

Actual values for these results are shown in Table 4. From factor to factor, the means for times to download the files, and their standard deviations, are very similar.

4.2.2 Host Decryption Results. Since there are only two configurations to compare in this experiment (with VMI and without VMI), the authors used a t-test to compare the data sets. Using the lack of VMI as the population, the null hypothesis is that with VMI enabled, the observational data shall not be significantly different than without VMI. In accordance with the factorial design in the section above, 30 trials were run with and without VMI, reading and writing files of 1MB, 10MB, and 100MB. Results are shown in Table 4 for file writes; file read results are shown in Table 5.

The samples collected from each of the tests were proved to follow normal distributions using Q-in-Q plots; the F-test was then run to determine equality of variance between the sets (e.g, no-VMI 1MB to with-VMI 1MB, etc.), using an $\alpha = 0.05$. Equality of variance was shown between all corresponding sets. The resultant t-test p-values for each of the set-pairs were:

- Write 1MB: $p = 0.914046758$
- Write 10MB: $p = 0.646450549$
- Write 100MB: $p = 0.308094339$
- Read 1MB: $p = 0.313688073$
- Read 10MB: $p = 0.317179742$
- Read 100MB: $p = 0.162026589$

Considering all of the p-values being higher than a 0.05 significance, we can accept the null hypothesis that there is no significant difference between file reading and writing with or without VMI running.

5 SUGGESTIONS FOR MITIGATION

The methodology and experiments herein show how a CSP can transparently eavesdrop on a VM through the hypervisor and extract private information or data from a perceived secure cloud environment. Although such a data privacy compromise may appear to be unavoidable, the authors have also researched ways to decrease the risk or mitigate the full impact. However, the mitigation approaches must be implemented by the CSP. No mitigation approach is currently available that can be implemented solely by the cloud customer.

AMD has recently introduced a new technology called Secure Encrypted Virtualization (SEV) and SEV with Encrypted State (SEV-ES)[19], which provide the ability to hide memory and (with SEV-ES) most registers from the hypervisor, treating the hypervisor as possibly malicious. SEV is somewhat easy to turn on, but SEV-ES requires support from the virtualized OS.

The architecture of the hypervisor can also make things more difficult for attackers. Take Microsoft's Hyper-V or Azure as an example. In this architecture, the hypervisor is booted before the root partition, which is essentially a VM where the managing operating system resides. The root partition's view of system memory can be managed by Hyper-V, which uses this vantage point to protect its own code from view of the root partition. Even a malicious administrator would have a much more difficult time getting any code to run in the context of the hypervisor with this architecture. The hypervisor can utilize technologies such as Virtualization Technology for Directed I/O (VT-d) to prevent direct memory access (DMA), as well as EPT to block normal access to its own memory. Without these basic protections in place (for example with KVM and some others), it is trivial to run a kernel module in the host operating system which then locates the VM-Exit handler and hooks execution (as the authors have shown). But with these protections available and enabled, an attacker must exploit a vulnerability to gain access to the execution path of the hypervisor.

In addition to making the hypervisor's memory unavailable to at least typical access, it would also be useful if EPT and VT-d were used to protect the memory assigned to each guest. As noted, the LSASS attack should require nothing but direct access to host memory. The only way to prevent this would be to segment off memory from each guest. As to the authors' knowledge, this is currently not done by default in any hypervisor, although Hyper-V clearly has the ability as they have introduced Virtual Trust Levels (VTLs), which do protect the memory of high VTL guests from those with lower trust levels. This is currently used to run a secure VM (secure kernel) with full access to a Hyper-V enlightened Windows operating system and allow for it to perform PatchGuard-like activities [5] from beyond the purview of that operating system. It is reasonable to assume that this is also extendable to protect all guests from the already-untrusted Windows partition.

6 CONCLUSIONS

In this paper, the authors have presented several inherent risks to the protection of user data when hosted by IaaS-based platforms. The paper described how network and disk encryption in the cloud may not be sufficient to protect the confidentiality, integrity and privacy of user data. The authors described and demonstrated that a hypervisor agnostic monitoring tool can transparently access information, in a real-time fashion without the tenant's knowledge and with negligible performance impact to the tenant. Finally, the discussion covered some mitigations that could be put into place by the CSP to provide additional security to the users.

Table 3: Network Experiment Results (All times in milliseconds.)

No-VMI*	1KB	500KB	1MB	VMI-FF*	1KB	500KB	1MB	VMI-L*	1KB	500KB	1MB
Mean	7.99	17.19	67.17	Mean	8.85	17.12	66.81	Mean	7.54	17.00	68.32
Std Dev	1.95	2.28	3.21	Std Dev	2.07	1.91	2.94	Std Dev	1.79	1.88	2.43
No-VMI+	**1KB**	**500KB**	**1MB**	**VMI-FF+**	**1KB**	**500KB**	**1MB**	**VMI-L+**	**1KB**	**500KB**	**1MB**
Mean	8.34	17.42	68.77	Mean	8.01	17.21	67.80	Mean	9.00	17.03	69.24
Std Dev	1.97	2.38	2.98	Std Dev	2.30	1.99	3.01	Std Dev	1.55	2.31	2.33

Table 4: Host Test Results: File Write

No-VMI	1 MB (ms)	10 MB (ms)	100 MB (ms)
Mean	8.50	12.167	75.58
Std Dev	2.77	2.68	4.75
VMI	**1 MB (ms)**	**10 MB (ms)**	**100 MB (ms)**
Mean	8.44	11.79	76.97
Std Dev	2.77	3.42	5.48

Table 5: Host Test Results: File Reads

No-VMI	1 MB (ms)	10 MB (ms)	100 MB (ms)
Mean	29.78	71.37	653.14
Std Dev	5.79	7.47	15.71
VMI	**1 MB (ms)**	**10 MB (ms)**	**100 MB (ms)**
Mean	28.18	73.24	660.14
Std Dev	6.14	6.59	21.45

REFERENCES

[1] Saad Alqahtany, Nathan Clarke, Steven Furnell, and Christoph Reich. Cloud forensics: a review of challenges, solutions and open problems. In *Cloud Computing (ICCC), 2015 International Conference on*. IEEE, 1–9.

[2] Anon. 2018. Virtual Machine Introspection. (2018).

[3] AWS and Jeff Barr. 2014. Use Your own Encryption Keys with S3's Server-Side Encryption. (2014). https://aws.amazon.com/blogs/aws/s3-encryption-with-your-keys/

[4] Micah Bushouse and Douglas Reeves. 2018. Goalkeeper: Comprehensive process enforcement from the hypervisor. *Computers & Security* 73 (2018), 459–473.

[5] Microsoft Corporation. 2006. *White Paper: Kernel Enhancements for Windows Vista and Windows Server 2008.* Technical Report. Redmond, WA, United States.

[6] Terri Moon Cronk. 2017. Defense Department to Move to Cloud Computing. (2017). https://www.defense.gov/News/Article/Article/1402556/defense-department-to-move-to-cloud-computing/

[7] ESXsi. 2016. NSX Manager Guest Introspection. (2016). https://esxsi.com/2016/09/28/guest-introspection/

[8] FireEye. 2018. Network Security: SSL Intercept. (2018). https://www.fireeye.com/products/nx-network-security-products/ssl-intercept.html

[9] Volatility Foundation. 2018. Volatility Foundation. (2018). http://www.volatilityfoundation.org/

[10] Yangchun Fu and Zhiqiang Lin. 2013. Bridging the semantic gap in virtual machine introspection via online kernel data redirection. *ACM Transactions on Information and System Security (TISSEC)* 16, 2 (2013), 7.

[11] Tal Garfinkel and Mendel Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Ndss*, Vol. 3. 191–206.

[12] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.Js: A Remote Software-Induced Fault Attack in JavaScript. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721 (DIMVA 2016)*. Springer-Verlag, Berlin, Heidelberg, 300–321. https://doi.org/10.1007/978-3-319-40667-1_15

[13] Yacine Hebbal, Sylvie Laniepce, and Jean-Marc Menaud. Virtual machine introspection: Techniques and applications. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*. IEEE, 676–685.

[14] Jennia Hizver and Tzi-cker Chiueh. 2014. Real-time deep virtual machine introspection and its applications. *SIGPLAN Not.* 49, 7 (2014), 3–14. https://doi.org/10.1145/2674025.2576196

[15] IBM. 2018. Privacy on the IBM Cloud. (2018). https://www.ibm.com/cloud/privacy

[16] Wayne Jansen and Timothy Grance. 2011. *SP 800-144. Guidelines on Security and Privacy in Public Cloud Computing.* Technical Report. Gaithersburg, MD, United States.

[17] Salman Javaid, Aleksandar Zoranic, Irfan Ahmed, and Golden G Richard III. Atomizer: Fast, Scalable and Lightweight Heap Analyzer for Virtual Machines in a Cloud Environment. In *6 th Layered Assurance Workshop*. 57.

[18] Jacob M Kambic. 2016. Extracting CNG TLS/SSL artifacts from LSASS memory. (2016).

[19] David Kaplan. 2017. *White Paper: Protecting VM Register State with SEV-ES.* Technical Report. Santa Clara, CA, United States.

[20] Frank Konkel. 2018. Defense Agency To Begin Moving Classified Data to Amazon's Secret Cloud After Protest. (2018). https://www.nextgov.com/it-modernization/2018/03/defense-agency-begin-moving-classified-data-amazons-secret-cloud-after-protest/146619/

[21] Tamas K Lengyel, Justin Neumann, Steve Maresca, Bryan D Payne, and Aggelos Kiayias. Virtual Machine Introspection in a Hybrid Honeypot Architecture. In *CSET*.

[22] Lionel Litty, H Andrés Lagar-Cavilla, and David Lie. Hypervisor Support for Identifying Covertly Executing Binaries. In *USENIX Security Symposium*. 243–258.

[23] Carsten Maartmann-Moe, Steffen E Thorkildsen, and André Årnes. 2009. The persistence of memory: Forensic identification and extraction of cryptographic keys. *digital investigation* 6 (2009), S132–S140.

[24] Sheik Khadar Ahmad Manoj and D Lalitha Bhaskari. 2016. Cloud Forensics-A Framework for investigating Cyber Attacks in cloud environment. *Procedia Computer Science* 85 (2016), 149–154.

[25] minimega. 2018. minimega: a distributed VM management tool. (2018). http://minimega.org/

[26] Mozilla, Peter Wu, and et al. 2018. NSS Key Log Format. (2018). https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

[27] Onur Mutlu. 2017. The RowHammer Problem and Other Issues We May Face As Memory Becomes Denser. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE '17)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 1116–1121. http://dl.acm.org/citation.cfm?id=3130379.3130643

[28] NIST and Emmanuel Aroms. 2012. *NIST Special Publication 800-53 Revision 3 Recommended Security Controls for Federal Information Systems and Organizations.* CreateSpace, Paramount, CA.

[29] NIST, Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. 2012. *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292).* CreateSpace Independent Publishing Platform, USA.

[30] NIST, Peter M. Mell, and Timothy Grance. 2011. *SP 800-145. The NIST Definition of Cloud Computing.* Technical Report. Gaithersburg, MD, United States.

[31] Graz University of Technology. 2018. Meltdown and Spectre. (2018). https://meltdownattack.com/

[32] OWASP. 2015. Man-in-the-middle attack. (2015). https://www.owasp.org/index.php/Man-in-the-middle_attack

[33] Bryan D Payne. 2012. Simplifying virtual machine introspection using libvmi. *Sandia report* (2012), 43–44.

[34] Bryan D Payne, DP de A Martim, and Wenke Lee. Secure and flexible monitoring of virtual machines. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 385–397.

[35] Jonas Pfoh, Christian Schneider, and Claudia Eckert. Nitro: Hardware-based system call tracing for virtual machines. In *International Workshop on Security*. Springer, 96–112.

[36] Rekall. 2018. Rekall Forensics. (2018). http://www.rekall-forensic.com/
[37] Vassil Roussev, Irfan Ahmed, Andres Barreto, Shane McCulley, and Vivek Shanmughan. 2016. Cloud forensicsâĂŞTool development studies & future outlook. *Digital investigation* 18 (2016), 79–95.
[38] Manuel Sández and Jesús Rivas. 2015. A review of technical problems when conducting an investigation in cloud based environments. *arXiv preprint arXiv:1508.01053* (2015).
[39] John Saxon, Behzad Bordbar, and Keith Harrison. 2015. Introspecting for RSA Key Material to Assist Intrusion Detection. *IEEE Cloud Computing* 2, 5 (2015), 30–38.
[40] John T Saxon, Behzad Bordbar, and Keith Harrison. Efficient Retrieval of Key Material for Inspecting Potentially Malicious Traffic in the Cloud. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 155–164.
[41] Mathieu Tarral. 2018. KVM Virtual Machine Introspection. (2018). https://github.com/KVM-VMI
[42] VMware. 2018. VMware vSockets Documentation. (2018). https://www.vmware.com/support/developer/vmci-sdk/
[43] Chonghua Wang, Zhiyu Hao, and Xiaochun Yun. NOR: Towards Non-intrusive, Real-Time and OS-agnostic Introspection for Virtual Machines in Cloud Environment. In *International Conference on Information Security and Cryptology*. Springer, 500–517.
[44] Hyun wook Baek, Abhinav Srivastava, and Jacobus Van der Merwe. Cloudvmi: Virtual machine introspection as a cloud service. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 153–158.
[45] Rui Wu, Ping Chen, Peng Liu, and Bing Mao. System call redirection: A practical approach to meeting real-world virtual machine introspection needs. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 574–585.