ARTICLE IN PRESS

COMPUTERS & SECURITY ■■ (2018) ■■–■■

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

GEMINI: Guest-transparent honey files via hypervisor-level access redirection

Zhongshu Gu [a,*], *Brendan Saltaformaggio* [b], *Xiangyu Zhang* [c], *Dongyan Xu* [c]

[a] *IBM T.J. Watson Research Center, Yorktown Heights, NY, USA*
[b] *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA*
[c] *Department of Computer Science and CERIAS, Purdue University, West Lafayette, IN, USA*

ARTICLE INFO

Article history:
Received 18 August 2017
Received in revised form 27 November 2017
Accepted 19 February 2018
Available online

Keywords:
Data security
System security
Virtualization
Access control
Moving target defense

ABSTRACT

Data safety has become a critical problem in the face of various cyber-attacks aiming at stealing or divulging sensitive information. In the event that adversaries have gained access to a system storing classified data, such crucial systems should actively protect the integrity of this data. To purposely deceive an attacker, we propose that accesses to sensitive data can be *dynamically partitioned* to prevent malicious tampering. In this paper, we present GEMINI, a virtualization-based system to transparently redirect accesses to classified files based on the context of the access (e.g., process, user, time-of-day, etc.). If an access violates preconfigured data-use policies then it will be rerouted to a *honey* version of the file, specifically crafted to be manipulated by the adversary. Thus, GEMINI transforms static, sensitive files into moving targets and provides strong transparency and tamper-resistance as it is located at the hypervisor level. Our evaluation shows that GEMINI effectively neutralizes several real-world attacks on various sensitive files and can be integrated seamlessly into current cloud environments.

1. Introduction

Data theft and leakage have become a crucial threat to cybersecurity (Messmer, 2014a, 2014b; Whittaker, 2012). Despite major advances in access control mechanism, adversaries still exploit the vulnerabilities in a system to implant software-backdoors, launch remote zero-day attacks, and access classified data. Unauthorized users with access to a high profile system are capable of exfiltrating sensitive digital information which may leak to the public in the future. State-of-the-art defense mechanisms try to address this problem from the perspective of eliminating system vulnerabilities, restricting users' privileges, and detecting attacks through anomaly analysis. Such approaches are typically effective for existing known attacks, but face many challenges when encountering new attacks. Furthermore, existing attacks may evolve to deviate from the detection mechanisms if they are well-studied by the adversaries.

In contrast to focusing on neutralizing all attacks, we aim to reinforce the protection on the victim, i.e., the sensitive data. The static nature of stored data files makes them easy to be studied and predicted by the adversaries. For example, a social engineering attack intending to steal a user's login information may create a phishing *Facebook* login page and redirect access to www.facebook.com to their fake site's IP address by

* *Corresponding author.*
E-mail addresses: zgu@us.ibm.com (Z. Gu), brendan@ece.gatech.edu (B. Saltaformaggio), xyzhang@cs.purdue.edu (X. Zhang), dxu@cs.purdue.edu (D. Xu).
https://doi.org/10.1016/j.cose.2018.02.014
0167-4048/

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

rewriting the `/etc/hosts` file (the standard DNS translation local store) on the victim's computer.

To address such attacks, we propose transforming the static, sensitive data file into a moving target when facing malicious tampering, i.e., we dynamically partition the accesses to sensitive data at runtime. Rather than maintaining only one copy of these valuable files, an administrator should prepare *honey* duplicates – versions of the files in which all sensitive content has been eliminated. Any future read or write of a valuable file can then be rerouted to access the original version or the honey version based on current context, e.g., user id, process type, and time-of-day. Therefore, an attacker with access to a system cannot retrieve the real data from the sensitive file, and malicious file modifications can be contained to the honey version, thus keeping the original file intact and avoiding the compromise.

Furthermore, we also notice recent research efforts (Bowen et al., 2009; Juels and Rivest, 2013; Park and Stolfo, 2012) on using decoy resources for detecting security breaches. Honey data (e.g., fake credit card number, decoy credential, etc.) contained in the decoy files embed distinctive signatures. If the attackers use the honey data retrieved from the decoy resources, their footprints will be accurately identified and traced by the intrusion detection system. To integrate with their approaches, we can redirect illegal accesses (which fail security checks) to retrieve the honey data from the decoy files, which can signal an alert when they are used.

We have developed Gemini, a dynamic delegation system supporting file access redirection within virtual machines (VM) to provide sensitive file protection. Gemini allows system administrators to define fine-grained security policies paired with specially crafted honey files to perform file access redirection. Gemini is implemented at the hypervisor level, thus it is transparent to the guest execution environment, i.e., both the guest operating system and the applications running within it, and free from tampering of attacks within the guest VM.

The rest of the paper is organized as follows. Section 2 presents the key idea and assumptions of Gemini. Section 3 gives a detailed design of the whole system and presents some technical details of the implementation. Section 4 provides three case studies on the effectiveness of Gemini and shows the performance evaluation. Section 5 discusses the limitation of the current prototype and the future work. Section 6 describes related work and we conclude in Section 7.

2. System overview

2.1. *Key idea: File access delegation*

To achieve portability and backward compatibility of applications across similar operating system (OS) kernel versions, system call interfaces are generally consistent. For example, with the availability of *IEEE POSIX* standards, most Linux applications can run on all Unix-like platforms without modifying/recompiling their source code. Furthermore, for file operations in Unix-like systems, a series of file-IO system calls are provided and invocations of these functions which operate on the same file are connected via a file descriptor (i.e., returned from the open system call).

Based on the observation that file-IO system call interfaces are generally consistent, we develop the key technique behind Gemini: transparent delegation of file access from one system to another. To be more specific, we are able to reroute file accesses from the in-VM file system to a file system outside of the VM, which we denote as the *delegation target file system*.

To apply the technique on sensitive file protection, if some file we are monitoring is accessed by the file-IO system calls, we intercept this access before it reaches the real content of the file and check predefined security policies. If the access meets the security policies defined for this file, we allow it to manipulate the original file – in either the in-VM file system or the out-of-VM *delegation target file system*. If it violates any policy then we dispatch this system call to a honey version of this file and return the honey file's results back to the system call.

Security policies can be defined by system administrators based on their needs, especially to complement the default file permissions provided by the guest OS. Currently, we have defined three types of security policies and it is convenient to extend to more categories in the future.

A *User Policy* defines the users that are allowed to access a file. We check the user id of any process that performs a file access. If the process does not belong to some user id list that is defined in this user policy, then we redirect the access to a honey version of the file. This intends to provide finer-grained user privilege differentiation and retrofit existing in-VM permission checking on the sensitive files.

A *Process Policy* defines which processes are allowed to access the file. We can check the characteristics and integrity of processes that execute the file access. If that process does not belong to some list of processes that are defined in the process policy, we redirect the access to the honey version of the file. This policy constrains the spectrum of processes that can be used to access a classified file. This will effectively filter file accesses from atypical processes in the system (likely created by adversaries).

A *Time Policy* defines the time window in a single day that access to a file is granted. If the current time is within this time window, access to the file is allowed. Otherwise, the access is dispatched to the honey version. For example, the administrator can make time policy to allow accesses to the original file from 9AM to 5PM every day, but reroute them after work. In a more restricted case, a legitimate user can leverage this policy to further minimize the attacker's time window. The administrator can enable the time policy (by setting the start time to the current time) first. After the file usage has finished, he or she can then disable the time policy by marking the entire time window in this policy as "off-limits". Thus the only chance an adversary has to access the file is simultaneous with a legitimate user (within the same time window), which we consider unlikely in practice.

2.2. *Assumptions*

We assume that the semantics of file-IO system call interfaces (including the parameters and the return value) to manipulate files within the guest VM and the *delegation target file system* are the same. For example, we currently do not yet support a scenario such as a Linux VM and a Windows

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

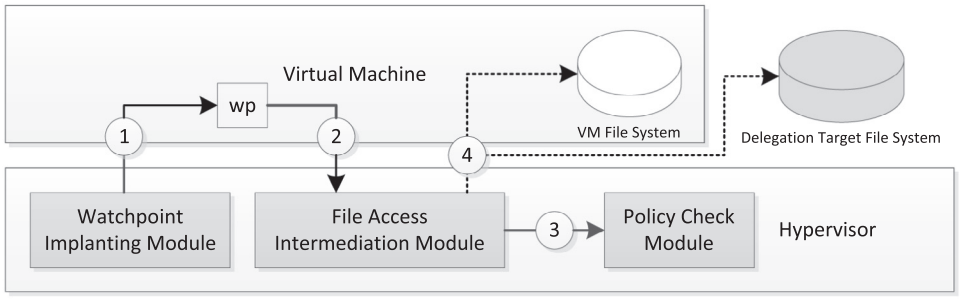

Fig. 1 – The design of GEMINI. (1) Implant watchpoints into virtual machine. (2) Handle watchpoint event. (3) Check security policy. (4) Dispatch requests based on Step 3.

delegation target system because the system call interfaces are not compatible. In practice, this assumption is quite reasonable because most Unix-like systems have the same file-IO system call interfaces for maintaining backward compatibility.

3. Design and implementation

The GEMINI system consists of three modules located in the hypervisor: Watchpoint Implanting Module, File Access Intermediation Module, and Policy Check Module. Fig. 1 illustrates the interactions between these modules and the basic workflow of GEMINI. We give a detailed description of each module's functionality in this section.

3.1. *Watchpoint implanting module*

Applications access files through file-IO system calls. Before booting the protected guest system, we implant watchpoints at the entry addresses of file-IO system calls.[1] Each of these watchpoints will trigger a VM exit event to be intercepted by the hypervisor.

Considering that the loading addresses for these system calls should not be the same across different OS kernel versions, we cannot simply set watchpoints at fixed addresses. Instead, we first inspect the virtual machine disk image using the `Network Block Device` (NBD) protocol, mount the nbd node on the host file system, and retrieve the corresponding system call addresses from its `System.map` (in the guest's /boot directory).

3.2. *File access intermediation module*

We then intercept any VM exits caused by the watchpoints in the *File Access Intermediation Module*. Based on the program counter of current VM exit, we execute an event handler corresponding to the file-IO system call that the guest was executing. Each event handler first checks the requested file's path to verify whether this file that we are monitoring is a sensitive file. It then checks the security policies assigned to that file to determine whether this file access meets the redirection conditions. If the file access violates any policy's requirements, then GEMINI marks this file access for redirection to the honey version. If the file access does not violate any policies on the target file, then we simply allow the access to the original file.

To redirect the file accesses, we generate the same system call on the corresponding file in the *delegation target file system*.[2] After the system call on the delegation target system returns, we set the instruction pointer in the guest VM to the return address of the original system call (i.e., to skip the function body of this system call) and set the return value to what the delegation target system returned.

There are some technical challenges worth mentioning during our implementation of GEMINI and we give detailed descriptions of our solutions in the following.

3.2.1. *Conflict of file descriptor*

Before operating on any file, a process must first obtain a file descriptor – an integer value returned from the open system call. Subsequent file-IO system calls use this file descriptor to perform operations on the file. File descriptors are bound to the process context, and in order to avoid a conflict of file descriptors, GEMINI cannot simply return a file descriptor from the delegated open system call to the guest VM.

To address this problem, GEMINI creates a special system-wide (in contrast to the process-wide integer space for each process within VM) integer space for sensitive file descriptors, which has no conflicts with the standard file descriptor integers. Upon returning from the open system call, GEMINI returns a file descriptor within the bounds of this special integer space, and internally notes the mapping between the special file descriptor and the file being opened. When another system call tries to manipulate the file using a file descriptor in this integer space, we can distinguish it instantly from other standard file descriptors.

3.2.2. *Enforced mapping on unmapped buffer*

In order to determine whether GEMINI needs to delegate a system call, it needs to interpret the system call's parameters to check the conditions in the security policy. Integer parameters are available instantly when you intercept the system call, but for

[1] sys_{open, close, read, write, creat, access, fstat64, lstat64, llseek, mmap_pgoff, munmap, fcntl64, mprotect}

[2] System administrators can decide where to store the honey file, either in *delegation target file system* or in VM file system. Please see more discussion in Section 5.2.

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

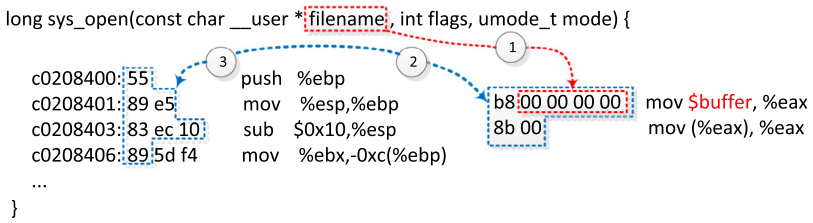

Fig. 2 – Enforced mapping on unmapped buffer. (1) Load filename's memory address into $buffer. (2) Replace the first 7-byte instructions of the function body with mov $buffer,%eax;mov (%eax),%eax. (3) Rewrite the original instructions back and re-execute them.

parameters which are buffer pointers, due to lazy loading in demand paging, the memory pages referenced by this buffer pointer may not be readily mapped in the process' page table (they will be finally mapped when the buffer is accessed later, triggering a page fault in the VM). This situation makes it impossible to retrieve the content of the buffer from our intercept point (i.e., the entry point of the system call). Because some pointers in the parameters are critical for our policy checking, e.g., filename in the `sys_open`, Gemini simulates the memory read/write instructions on the unmapped buffer and force the virtual machine to fix the page faults. This causes the VM to naturally load the referenced memory pages and upon the page fault handler's return, Gemini can access the buffer contents it needs.

Fig. 2 shows a concrete example of simulating an in-VM memory read to cause a page fault. First, Gemini prepares two instructions (shown in AT&T style), `mov $buffer,%eax; mov (%eax),%eax`, in which `$buffer` is preset as `00 00 00 00`. Their purpose is to read memory from `$buffer` – which is unmapped at this point. Step 1 reads the value of the pointer `filename` and loads it into `$buffer` when `sys_open` is intercepted. Then Gemini rewrites the first 7 bytes (`55 89 e5 83 ec 10 89` in this case) of the original `sys_open` function body with the prepared instructions and resumes the VM execution in Step 2. After these two instructions are executed, the guest VM fixes the page fault on `filename` and Gemini again intercepts the execution. The original 7-byte instructions are written back and the instruction pointer is set to the first instruction in `sys_open` in Step 3. At this time, `filename`'s memory is mapped and its content is available for retrieving.

The simulation of a memory write is similarly performed by swapping the operands in the second instruction. For example, we leverage a memory write to trigger a page fault on the `buf` parameter of `sys_read`, which will be filled with contents read from the file.

3.2.3. *Memory mapped files*

In modern Unix-like systems, there are basically two ways to read/write a file. The first way is to use `sys_read/sys_write` to manipulate the file directly. The other way is to mmap the file into memory and read/write the memory. For the former, because subsequent file operations also leverage system calls, we are able to intercept each operation during the life-cycle of manipulating the file. For the latter approach, the memory mapped region (returned by mmap system call) for any file in the *delegation target file system* will not be present in the process' address space in the virtual machine (as it belongs to the delegation target machine).

In order to address this problem, Gemini first allocates additional physical memory from the hypervisor, and the requested file is mapped into these new memory pages. Gemini then searches for empty memory holes in the application's virtual address space and when a large enough hole is found, creates the virtual-to-physical address mapping for this memory region by adjusting the in-VM page table. This ensures that any subsequent memory reads/writes to this area are already directed at the mapped file from the *delegation target file system*.

3.3. *Policy check module*

Policy rules are defined by the system administrators and can be extended to check virtually any combination of data available via virtual machine introspection. Currently, we have three types of security policies as we mentioned in Section 2. Because the policy checking is performed at the hypervisor level, we leverage traditional virtual machine introspection techniques to extract any needed semantic information, e.g., user id, process type, etc., from the memory of the virtual machine. Furthermore, the policies are tamper-resistant to any attacks within the virtual machine and can be enabled, disabled, or modified dynamically at runtime.

3.4. *Prototype implementation*

We integrate our Gemini prototype with the KVM hypervisor (Kivity et al., 2007) (i.e., kvm-kmod-3.6 and qemu-kvm-1.2.0). The host (delegation target) system is Linux Mint 13 x86_64 (Linux Kernel version 3.5.0) and the guest VM runs Ubuntu 10.04 (Linux Kernel version 2.6.32) i386 LTS release. In total, we add 1867 SLOC in C for the development of Gemini.

4. Evaluation

In this section we present the evaluation of our system in two aspects: security and performance. For the security evaluation, we demonstrate the efficacy of our system for providing protection to sensitive files. For the performance evaluation, we compare the performance results before enabling Gemini with after enabling Gemini to demonstrate the feasibility of deployment. The hardware configuration of our testing platform is a Lenovo Ideapad U410 with Intel® Core™ i7 3.10 GHz and 8 GB

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

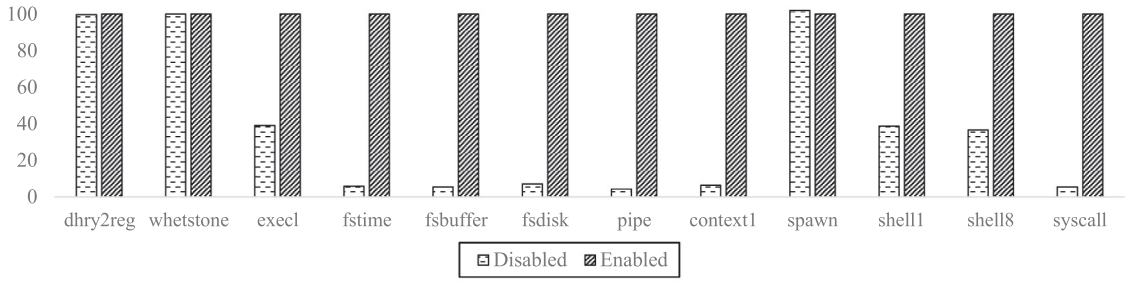

Fig. 3 – Normalized index by UnixBench Benchmark Group.

memory. We allocate 2 GB memory for the guest VM, which runs Ubuntu 10.04 (Linux kernel version 2.6.32) i386 LTS release.

4.1. *Security evaluation*

4.1.1. *Case study 1: Gailly, 2002*

John the Ripper is one of the most popular password cracker tools as it can run on fifteen different platforms, including both Unix-like systems and Windows. Further, it can be used against many popular password formats, including several crypt password hash types most commonly found in various flavors of Unix. We leverage *John the Ripper* to uncover weak passwords in `/etc/shadow`, which contains account information, the encrypted passwords, and expiration values.

Without enabling Gemini, weak passwords (e.g., those vulnerable to the dictionary attack) can be easily discovered by the cracker tool. At a high level, *John the Ripper* works as follows: it first generates encrypted strings (in the same format as the password being examined) from a dictionary or a list of commonly used password. It then compares the encrypted strings to the encrypted password stored in the system.

Thus, we aim to disarm this attack by presenting a honey version of `/etc/shadow` to the cracker tool. After enabling Gemini, we redirect file accesses to `/etc/shadow` to a version that contains no real user account and password. We enable a process policy that only allows legitimate processes, e.g., `passwd`, to access `/etc/shadow`. This successfully prevented *John the Ripper* from accessing the real `/etc/shadow` file because the cracker tool is not allowed to access the original file. Therefore, `John the Ripper` produced no useful output because it cannot extract any real passwords.

Furthermore, we may fill the honey `/etc/shadow` with decoy login credentials. Any attacker that leverages the decoy information to log in can be accurately pinpointed by the intrusion detection system.

4.1.2. *Case study 2: Infelf (ZOMBiE/29A, 2002)*

Infelf is an offline binary infection tool that injects a malicious payload into an existing benign application binary to generate a trojaned binary. Its basic functionality is to split trojan code into multiple instruction blocks, insert them into free alignment areas between functions, and concatenate them with jump instructions. Thus it incurs no increase in the binary's size.

We use *Infelf* to implant a hardware register printing function into the *gvim* binary and redirect its entry function to this trojan code. Without enabling Gemini, the attack can successfully transform the binary into a trojaned application. To prevent this attack, we enable a process policy to redirect file access to a honey *gvim* binary if the current process is not in the whitelist for accessing *gvim*. In the presence of Gemini, *Infelf* can only infect the honey version of the *gvim* binary, and thanks to Gemini's redirection the original binary is left intact.

4.1.3. *Case study 3: Separate login password for different time window*

With the availability of Gemini, a system administrator can log in the same user with different passwords for different time windows. For example, during working-hours an employee may use one set of login credentials which are only valid from inside of the company's headquarters. Later, to work from outside of the office, another set of credentials must be used. To set up such protection, an administrator just needs to set the `/etc/passwd` and `/etc/shadow` files as Gemini monitored files, design two copies (i.e., the working-hours version and after-work version) for each file, and set up a time policy with the access time windows for both files. Afterwards, Gemini automatically handles the password switch dynamically according to the time policy. Therefore, even if adversaries retrieve the after-work password, they are not able to log in the system at working hours. Plus, the time policy can only be reset from the hypervisor level, and thus it is impossible for the adversaries to read or modify the time window from within VM.

4.2. *Performance evaluation*

We use *UnixBench* to evaluate whole system performance after enabling Gemini. By default, we install the user policy, process policy, and time policy that were mentioned in Section 3. However, compared to hypervisor-level interceptions of file-IO system calls, policy checking and policy enforcement contribute negligible addition to the whole system performance overhead. In Fig. 3, we normalize the performance index of *UnixBench* (higher performance index indicates better performance) and present a comparison for all sub-benchmarks. We find that, compared with the baseline result, we can maintain the same performance level for *dhry2reg*, *whetstone*, and *spawn*, which are indicative of computing-intensive tasks. We observe 60% performance overhead on *shell/execl* and 16× overhead on the sub-benchmarks involving intensive file-IO system calls. This is due to Gemini's interception on all file-IO system calls. From our testing, enabling Gemini does not affect the user experience for typical applications. This is because that, for a general application, file operations are less frequent than in

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

the *UnixBench* file-IO sub-benchmarks. Most computation on the file contents is conducted on the file buffers loaded in the main memory before flushing back to the disk. GEMINI does not incur performance overhead for operations in the main memory. Any performance degradation on file-IO operations will be amortized over the lifetime of the application.

5. Discussion

5.1. *Detection of GEMINI*

GEMINI is located in the hypervisor and is designed to be transparent to the guest VM, but it is still possible to detect its existence.

The first possible detection approach is a timing attack. By measuring the system call execution time to access files and compare with another similar virtual machine with GEMINI disabled, a program may be able to identify the difference. This timing evidence may be used to reason about the existence of GEMINI.

The other detection approach is to interpret the file descriptor integer returned from the open system call. As described in Section 3, in order to avoid conflicts with traditional file descriptors, GEMINI returns an integer within a special integer space. Assuming adversaries know the range of this integer space in advance, they may consider any file descriptor in that range as evidence of GEMINI.

5.2. *Bypassing GEMINI*

If an attack does not use traditional system calls to access files, then it will not be tracked by GEMINI, i.e., the attack may be able to read/write files within the VM file system using kernel function directly. But fortunately, it is impossible for adversaries to manipulate files in the out-of-VM *delegation target file system*. To address this, we recommend system administrator storing the sensitive files in the *delegation target file system* outside of VM and put the honey files inside the VM. Thus even if the VM is compromised by the attacker, the sensitive files are still safe.

In addition, for the current implementation of GEMINI, attackers may bypass file path checking by creating symbolic links to sensitive files. This weakness can be addressed by further modeling system calls related to symbolic links, such as `sys_symlink` and `sys_symlinkat`. We leave this as our future work of GEMINI.

6. Related work

GEMINI can be classified into the research of virtual machine introspection(VMI) technology. In this section, we review some representative works in VMI and compare them with GEMINI.

Various VMI techniques have been developed to retrofit the security of the system running within a virtual machine. These are based on the concept that introspection tools run one layer lower than the virtual machine and thus it is hard for adversaries to tamper with their integrity. We can basically classify existing VMI techniques into two categories: passive introspection and active introspection.

In the former category, the main functionality of VMI is to monitor the execution status and inspect the memory state. Livewire (Garfinkel and Rosenblum, 2003) is the pioneering work proposing VMI methodology to detect malware infections. Following it, XenAccess (Payne et al., 2007), VMwatcher (Jiang et al., 2007), VMscope (Jiang and Wang, 2007), Antfarm (Jones et al., 2006), and Ether (Dinaburg et al., 2008) are some representative "out-of-the-box" research efforts to monitor the guest VM at the hypervisor level.

One of the most well-known challenges of VMI is the semantic gap (Chen and Noble, 2001). From the view of out-of-VM introspection tools, the whole virtual machine's state is a black box. We need to reconstruct the semantic view of the guest VM with only limited information exposed to the hypervisor. In order to automatically bridge the semantic gap, Virtuoso (Dolan-Gavitt et al., 2011) made some initial efforts to collect training traces from in-VM tools and automatically translate them into out-of-VM introspection programs running at the hypervisor level. VMST (Fu and Lin, 2012) is another research effort on bridging semantic gap. The key idea of VMST is to reuse the introspection tool's code in a trusted VM and redirect data accesses to the VM that needs monitoring.

Compared with passive monitoring, active introspection approaches are driven by events within the virtual machine and they perform interference on the runtime execution. IntroVirt (Joshi et al., 2005) executes vulnerability-specific predicates in a VM for intrusion reproduction. Lycosid (Jones et al., 2008) uses cross-view validation techniques to detect maliciously hidden OS processes and patches the executable code to influence the process runtime. Manitou (Litty and Lie, 2006) detects corrupted instruction pages by comparing their hashes with memory-page hashes at runtime and marks them non-executable to eliminate attacks. Lares (Payne et al., 2008) and SIM (Sharif et al., 2009) propose active monitoring from outside the untrusted VM. Hooks are implanted inside the guest VM to track the executing events and invoke the security tool responsively. Lares places the security tool in another trusted virtual machine and lets the hooked events trigger a virtual machine switch. On the contrary, SIM creates a separate guest address space to gain an in-context view and native speed. Process Implanting (Gu et al., 2011) shows that it is also feasible to inject an introspection process from the hypervisor into a guest VM (running under the cover of an existing process context) to gain both the in-VM context and out-of-VM protection. FACE-CHANGE (Gu et al., 2014) is a virtualization-based approach that controls dynamic kernel switching to minimize attack surface for each individual process. Hypershell (Fu et al., 2014) and ShadowContext (Wu et al., 2014) are two recent research efforts further investigating the idea of system call redirection and process injection in a virtual machine.

GEMINI can also be classified as an active introspection approach. It reacts directly to file-IO system call events and reconstructs the semantics for file operations in the hypervisor level. We share some concepts with previous active introspection approaches and push it further to a new security area for sensitive file access delegation.

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

7. Conclusion

In this paper, we present Gemini, a virtualization-based dynamic delegation system supporting sensitive file access redirection. By enabling Gemini, system administrators can design rules for redirecting file access on sensitive files to honey versions specifically crafted for adversaries. Thus Gemini transforms static sensitive files into moving targets for the attackers. From our evaluation, we demonstrate the effectiveness of Gemini on neutralizing various cyber-attacks that manipulate critical system files and its practicality of deploying this technique in the existing cloud environment.

REFERENCES

Bowen BM, Hershkop S, Keromytis AD, Stolfo SJ. Baiting inside attackers using decoy documents. Springer; 2009.

Chen P, Noble B. When virtual is better than real, in: HOTOS, Published by the IEEE Computer Society, 2001, p. 0133.

Dinaburg A, Royal P, Sharif M, Lee W. Ether: malware analysis via hardware virtualization extensions, in: Proceedings of the 15th ACM conference on Computer and communications security, ACM, 2008, pp. 51–62.

Dolan-Gavitt B, Leek T, Zhivich M, Giffin J, Lee W. Virtuoso: narrowing the semantic gap in virtual machine introspection, in: Security and Privacy (SP), 2011 IEEE Symposium on, IEEE, 2011, pp. 297–312.

Fu Y, Lin Z. Space traveling across VM: automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection, in: Security and Privacy (SP), 2012 IEEE Symposium on, IEEE, 2012, pp. 586–600.

Fu Y, Zeng J, Lin Z. HyperShell: a practical hypervisor layer guest OS shell for automated in-VM management, in: Proceedings of the 2014 USENIX Annual Technical Conference, Philadephia, PA, 2014.

Gailly J-L. John the Ripper password cracker; 2002. http://www.openwall.com/john/. [Accessed 10 March 2018].

Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection, in: Proc. Network and Distributed Systems Security Symposium, Vol. 1, Citeseer, 2003, pp. 253–85.

Gu Z, Deng Z, Xu D, Jiang X. Process implanting: a new active introspection framework for virtualization, in: Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on, IEEE, 2011, pp. 147–56.

Gu Z, Saltaformaggio B, Zhang X, Xu D. Face-change: application-driven dynamic kernel view switching in a virtual machine, in: Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on, IEEE, 2014, pp. 491–502.

Jiang X, Wang X. Out-of-the-box monitoring of VM-based high-interaction honeypots, in: Proceedings of the 10th international conference on Recent advances in intrusion detection, Springer-Verlag, 2007, pp. 198–218.

Jiang X, Wang X, Xu D. Stealthy malware detection through VMM-based out-of-the-box semantic view reconstruction, in: Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 128–38.

Jones S, Arpaci-Dusseau A, Arpaci-Dusseau R. Antfarm: Tracking processes in a virtual machine environment, in: Proceedings of the USENIX Annual Technical Conference, 2006, pp. 1–14.

Jones ST, Arpaci-Dusseau AC, Arpaci-Dusseau RH. VMM-based hidden process detection and identification using Lycosid, in: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, ACM, 2008, pp. 91–100.

Joshi A, King S, Dunlap G, Chen P. Detecting past and present intrusions through vulnerability-specific predicates, in: Proceedings of the twentieth ACM symposium on Operating systems principles, ACM, 2005, pp. 91–104.

Juels A, Rivest RL. Honeywords: making password-cracking detectable, in: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 145–60.

Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: the Linux virtual machine monitor, in: Proceedings of the Linux Symposium, Vol. 1, 2007, pp. 225–30.

Litty L, Lie D. Manitou: a layer-below approach to fighting malware, in: Proceedings of the 1st workshop on Architectural and system support for improving software dependability, ACM, 2006, pp. 6–11.

Messmer E. The worst data breach incidents of 2013; Jan 2014a. Available from: http://www.networkworld.com/article/2286787/4g/135100-The-worst-data-breach-incidents-of-2013.html. [Accessed 10 March 2018].

Messmer E. The worst data breaches of 2014 … so far (q1); April 2014b. Available from: http://www.networkworld.com/article/2286300/malware-cybercrime/147526-The-worst-data-breaches-of-2014-so-far-Q1.html. [Accessed 10 March 2018].

Park Y, Stolfo SJ. Software decoys for insider threat, in: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ACM, 2012, pp. 93–4.

Payne B, Carbone M, Lee W. Secure and flexible monitoring of virtual machines, in: ACSAC, IEEE Computer Society, 2007, pp. 385–97.

Payne B, Carbone M, Sharif M, Lee W. Lares: an architecture for secure active monitoring using virtualization, in: Security and Privacy, 2008, IEEE Symposium on, IEEE, 2008, pp. 233–47.

Sharif M, Lee W, Cui W, Lanzi A. Secure in-VM monitoring using hardware virtualization, in: Proceedings of the 16th ACM conference on Computer and communications security, ACM, 2009, pp. 477–87.

Whittaker Z. Looking back at the major hacks, leaks and data breaches; 2012. Available from: http://www.zdnet.com/2012-looking-back-at-the-major-hacks-leaks-and-data-breaches-7000008854/. [Accessed 10 March 2018].

Wu R, Chen P, Liu P, Mao B. System call redirection: a practical approach to meeting real-world VMI needs, in: Proceedings of the 44th IEEE/IFIP International Conference on Dependable Systems and Networks(DSN 2014), Atlanta, PA, 2014.

Z0MBiE/29A. Infelf; 2002. http://z0mbie.daemonlab.org/infelf.html.[Accessed 10 March 2018].

Zhongshu Gu is a Research Staff Member in the Security Research Department of the IBM T.J. Watson Research Center. He received his Ph.D. from Purdue University in 2015 and B.S. from Fudan University in 2007, both in Computer Science. His research interests are in the areas of systems security, AI security, security analytics, and cyber forensics.

Brendan Saltaformaggio is an assistant professor in the School of Electrical and Computer Engineering at Georgia Tech. His research interests lie in computer systems security, cyber forensics, and the vetting of untrusted software. Originally from New Orleans, Dr. Saltaformaggio earned his Bachelor of Science with Honors in Computer Science from the University of New Orleans in 2012. He received his M.S. and Ph.D. in Computer Science at Purdue University in 2014 and 2016, respectively, during which

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014

Dr. Saltaformaggio was honored with the 2017 ACM SIGSAC Doctoral Dissertation Award.

Xiangyu Zhang is a professor and University Scholar at Purdue University. He works on dynamic and static program analysis and their applications in security, debugging, testing, and data processing. He has received the 2006 ACM SIGPLAN Distinguished Doctoral Dissertation Award, NSF Career Award, ACM SIGSOFT Distinguished Paper Awards, Best Student Paper Award on USENIX Security'14, Best Paper Award on CCS'15 and Distinguished Paper Awards on NDSS'16 and USENIX SECURITY'17.

Dongyan Xu is a professor of Computer Science at Purdue University. He is also the interim director of the Center for Education and Research in Information Assurance and Security (CERIAS). He has been on Purdue faculty since 2001, when he received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. His research efforts span computer systems security and forensics, cloud computing, and virtualization, with projects sponsored by both government agencies and industry. He is the co-author of seven award-winning papers at major conferences in security and cloud computing.

Please cite this article in press as: Zhongshu Gu, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu, GEMINI: Guest-transparent honey files via hypervisor-level access redirection, computers & security (2018), doi: 10.1016/j.cose.2018.02.014