2018 IEEE International Conference on Software Quality, Reliability and Security

Hypervisor-based Sensitive Data Leakage Detector

Shu-Hao Chang, Sanoop Mallissery, Chih-Hao Hsieh, Yu-Sung Wu
Institute of Computer Science and Engineering
National Chiao Tung University, Hsinchu, Taiwan
gakaza3333@gmail.com, sanoopmallissery@gmail.com, chhsieh0810@gmail.com, ysw@cs.nctu.edu.tw

***Abstract*—Sensitive Data Leakage (SDL) is a major issue faced by organizations due to increasing reliance on data-driven decision-making. Existing Data Leakage Prevention (DLP) solutions are being challenged by the adoption of network transport encryption and the presence of privileged-mode malware designed to tamper with the DLP agent programs. We propose a novel DLP system called "HyperSweep" that uses Virtual Machine Memory Introspection (VMI) technology to inspect the memory content of a guest system for sensitive information. The approach is robust against both network transport encryption and malware that attack DLP agent programs. The HyperSweep prototype is implemented on top of the KVM hypervisor. Our experiments have confirmed its applicability to real-world applications, including web browsers, office applications, and social networking applications. The experiments also indicate moderate performance overhead from applying HyperSweep.**

***Index Terms*—hypervisor, virtualization, virtual machine introspection, data security**

I. Introduction

Sensitive Data Leakage (SDL) is a major concern since the data used inside an organization could be vital information which weighs the competitive advantage and reputation of that organization. Nowadays many of the organizations outsource their data to a partner organization, and it may lead to data leakage while transferring which happens because of the inconsistent or improper security policies between the organizations. In enterprise computing environments, data leakage is traditionally managed by using network Data Leakage Prevention (DLP) mechanisms, which focuses on detecting the presence of sensitive data in the network traffic. However, its effectiveness is becoming limited by the adoption of network transport encryption. It cannot provide end-to-end protection for data transmitted through out-of-bound channels such as sharing data via USB thumb drives.

The deficiencies of network DLP solutions can be complemented by endpoint DLP mechanisms, which rely on pre-installed agent programs to track data access activities on host machines. However, the deployment and maintenance are more cumbersome than network DLPs due to the requirement of pre-installed agent programs. It also depends on the integrity of the endpoint Operating System (OS) to operate appropriately and limiting their effectiveness in the presence of malware or kernel rootkits.

In this work, a prototype of hypervisor-based endpoint DLP mechanism called HyperSweep has been proposed. The mechanism uses Virtual Machine Memory Introspection (VMI) technology [1] [2] [3] [4] to inspect the memory content of a guest system for the presence of sensitive data. This deployment is more transparent than the existing endpoint DLP mechanisms since the application workload can run unmodified in a guest Virtual Machine (VM). The use of virtualization provides inherent security isolation between HyperSweep and the guest system. By inspecting the memory content directly, the effectiveness of HyperSweep is unaffected by transport encryption.

In the proposed prototype of HyperSweep, the administrator can choose a variety of operation modes and scanning strategies which are explained later in Section IV. Hence the proposed HyperSweep provides flexibility for detection coverage and scanning speed. The proposed prototype is implemented on x86 KVM hypervisor, and it does not require any special hardware support. The experiment results confirm that Hyper- Sweep is capable of detecting the presence of sensitive data in the real-world application workload without degrading the performance of the host and guest system.

The rest of the paper is organized as follows: Section II describes the background study on data leakage prevention, virtual machine, and memory introspection. Section III focuses on the HyperSweep design, operation modes, and scanning strategies. Section IV discusses the implementation of the HyperSweep prototype followed with its evaluation in Section V. Section VI reviewed some of the related works, and Section VII concludes the work.

II. Background

The challenges on data security are never going to end, and it always risks the reputation of an organization. The existing endpoint DLP techniques are not able to overcome the vulnerabilities which may occur through the web applications, communication channels or maybe because of the misconfigured and outdated services which are used in a system. Hence protecting critical data is one of the crucial challenges and yet the information and network security teams are in trouble to avoid the complexity which occurs while deploying or managing data over the network. Many of the existing works on DLP focuses on the classification of data and detection of sensitive data [5] [6]. Some of the leading vendors of DLP solutions are Symantec, McAfee, Websense, CA Technologies, Palisade Systems, CA Technologies, NextLabs and Code Green Networks. The products released by these vendors are in line with the DLP techniques for file system monitoring, endpoint protection, e-mail and network monitoring. But most of the above mentioned DLP techniques did not satisfy the

978-1-5386-7757-5/18/$31.00 ©2018 IEEE
DOI 10.1109/QRS.2018.00029

end-user requirement concerning the sensitive data detection, complexity, and costs [7]. Also, the effectiveness of network-based DLP solutions is becoming limited due to increasing adoption of network transport encryption.

III. HyperSweep Design

HyperSweep is an endpoint DLP agent program designed to detect the presence of sensitive data in the memory space of target applications. HyperSweep adopts the hypervisor architecture, and hence the application workload will run unmodified in the guest VM. The HyperSweep agent identifies the sensitive information using the operation modes and scanning strategies which are explained in Section IV. HyperSweep runs on both XEN and KVM hypervisor and also works with Windows or Linux guest VM without altering the guest OS setting. The architecture of HyperSweep is

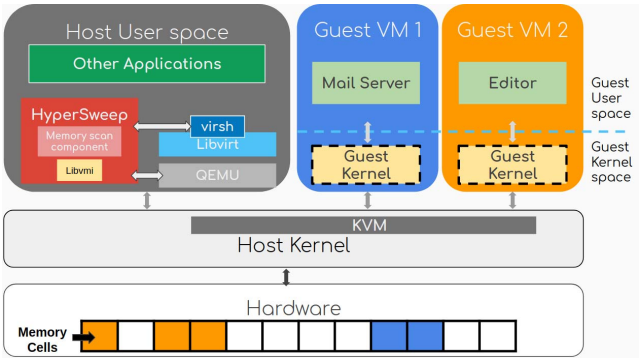

Fig. 1. HyperSweep overview

illustrated in Figure 1 and a typical environment setup is shown where multiple guest VMs will run their processes within each of the guest userspace. HyperSweep is running at host machines userspace where it will communicate with the XEN or KVM hypervisor and monitor these guest VMs. The administrator will hand out a list of sensitive information or data keywords that guest VM should not have gained access to the HyperSweep. To monitor the guest VMs, HyperSweep will have to go through multiple stages of mapping, checks the memory occupied by the guest VMs and trace through its memory structure.

HyperSweep will communicate and run along with Libvirt [8] and Quick Emulator (QEMU) [9] to manage the guest VM when it uses the KVM hypervisor. Libvirt is an open source API and management tool to manage VM and other virtualization functionality and technologies. QEMU is an emulator and virtualizer, which is responsible for performing hardware virtualization along with Libvirt and KVM to run VMs to achieve the performance almost as same as the native machine. To obtain the memory occupied by the guest VM, HyperSweep will rely on QEMU Machine Protocol (QMP) mechanism and issue commands to map guest VM to the host's physical memory. Hence the host will be able to monitor the memory that is operated by the guest VM. Further mapping detail is explained in Section V.

A. Operation Modes

HyperSweep supports two operation modes. In the *record-only mode*, HyperSweep only collects the memory dump for offline analysis, while in the *real-time mode*, HyperSweep provides online DLP protection. Details of the two operation modes are given in the following.

1) Record-only Mode: Record-only or Offline scanning is an old technique where HyperSweep scans a given memory dump file from the administrator and seek for sensitive information and keywords. Usually, the administrator will be looking at the physical memory of the guest VM because parsing through a physical memory dump file is more reliable, efficient, and least overhead method to detect the sensitive information. The kernel or userspace information of the guest VM will be stored within the physical memory dump file. Scanning for virtual memory based on the physical dump is still possible if the page table translation is found, and then HyperSweep will have to check whether the virtual memory section has been swapped out or not. However, most guest OS would swap out unused virtual memory due to limited assigned RAM size. Hence swapping out unused memory will optimize the memory usage performance. Therefore, scanning of virtual memory based on physical memory dump file often leads to an undesired result and hence not focused on the proposed design.

The offline scanning flow is shown in Figure 2. The memory scan component or agent of HyperSweep will parse through the physical memory dump file of the guest VM. Meanwhile, HyperSweep continually checks the configuration files and the sensitive information was given by the administrator to check for any violation.

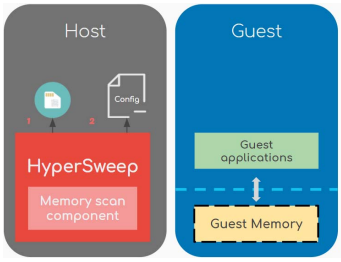

Fig. 2. HyperSweep with memory dump dcan

2) Real-time Mode: An administrator can decide to run the HyperSweep for a periodic scan or an on-demand scan on different guest VMs by considering the list of sensitive information or keywords. The administrator can issue an on-demand scan so that HyperSweep will immediately start a scan of the guest VM, or to have a periodic scan so the host machine will automatically scan the guest VMs after a period. By default, the possible overhead could cause HyperSweep to slow down and degrades the efficiency of running the guest VMs. Hence the periodic scan time-frequency has to be set carefully for the HyperSweep to balance with the overhead. On the other hand, the on-demand scan should run smoothly

without interfering the guest OS. In the meantime, a whitelist and blacklist can also be created to specify the processes or data information that wishes to be monitored.

In Real-time or online mode, HyperSweep operates with two scan strategies: the comprehensive scan and structural scan. Each scan strategy provides advantages to help HyperSweep. The comprehensive strategy scans the guest's memory thoroughly while structural strategy provides faster and more accurate scan on key areas of the memory. The detail of each scan strategies is explained in next section.

B. Scanning Strategies

Comprehensive or physical memory scan is an old strategy where the entire guest VMs physical memory to be traced to check the sensitive data. On the other hand, the structural scan is to trace through the OS processes memory structure and identify which process or processes are involved with the violation of sensitive data. However, offline scan mode does not gain access to guest VMs virtual memory. Combining both strategies, the administrator could get more information from structural scanning while receiving additional traces of evidence from comprehensive scanning.

With the previously mentioned modes and mechanisms provided, HyperSweep can run in periodic mode or perform an on-demand mode of the guest VMs physical or virtual memory. It depends on the administrator who needs to verify and act according to the situation.

1) Structural Scanning: The virtual structural scan is one of the primary focus of this proposed work, and it is required to obtain the information from the guest VM in real-time. The virtual structural scan will trace through the guest VM OS memory structure to acquire the detailed information of a particular fragment of guest VM memory. The detailed information includes assigned memory fragment length, start and end of heap or stack memory and number and size of memory assigned to particular processes. Hence, the focus in this work is mainly on the Linux memory structure. The guest OS type must be identified first, and then correct the memory structure variables for inspecting the guest VMs memory structure and hence to analyze for HyperSweep. HyperSweep needs to identify each guest process virtual memory, so categorizing each memory fragment is is essential. Therefore, a task descriptor or process descriptor becomes the starting point to identify and analyze the memory fragment content. Each process has its task descriptor, and all task descriptors are organized in a linked list. To further dig down into the memory structure, HyperSweep needs to gain access to the memory descriptor, or mm_struct, which is necessarily a summary of a programs memory stored in a doubly linked list. The memory descriptor mm_struct also contains VMA or vm_area_struct, an instance that entirely records the information of the memory interval or fragment.

Figure 3 shows the overview of the HyperSweeps memory scan component that accesses the guest VMs memory; the page directory, page table, and kernel data. First of all, the memory scan component must first confirm on the guest VM before checking with the system map for the memory structure. HyperSweep will read the copy of sysMap of the guest VM stored locally in the host machine, and any system map mismatch due to malware configuring the guest OS structure will be immediately noticed. HyperSweep then utilizes the system map to look for the virtual address for the kernel symbol to parse the guest OS structure. Hence after looking into the system map, HyperSweep maps the page directory to find the correct page table where the kernel data is stored. Therefore, it will be looking for the correct data page when the page table is mapped. Then the specific data page table or kernel data will be returned to the HyperSweep for sensitive data scan. Later, the memory scan component will try to map from Guest Virtual Address (GVA) to Guest Physical Address (GPA), so HyperSweep could map from GPA to Host Virtual Address (HVA) with QEMU before finally mapping to Host Physical Address (HPA).

2) Comprehensive Scanning: HyperSweep can perform the comprehensive scanning on the target guest VMs physical memory. Physical scanning is more reliable since data could still be stored in the physical memory block due to the lazy allocation for avoiding the loss of data. Another scenario is that the sensitive information may also be hidden in Virtual Memory Areas (VMAs) or because of a malware that intentionally tries to conceal from the guest OS. Therefore, HyperSweep could choose to parse through the physical memory instead of the virtual memory and possibly find traces of sensitive information that are left behind. The initial scan flow of comprehensive scanning is very similar to structural scanning shown in Figure 3. The only difference is that comprehensive scan path will not go beyond accessing the page directory since this is a physical memory scan instead of virtual memory scan. In other words, after checking the system map, the memory scan component will take the Guest Physical Address (GPA) and map directly to Host Virtual Address (HVA). From there, the hypervisor will map from Host Virtual Address (HVA) to Host Physical Address (HPA) to obtain the memory content stored within this location.

C. Pattern Matching Algorithms

The scanning algorithm is one aspect that influences the overall HyperSweep performance and efficiency at inspecting the guest VMs memory. There are many existing patterns matching algorithms, and HyperSweep adapted the well rounded Rabin-Karp search algorithm [10] for general search. The Rabin-Karp search algorithm uses hashing to find any one of a set of pattern strings in a text to perform the search. For the performance, a length of text 'n' and 'p' patterns with combined length 'm' have considered. The algorithm has an average and best case running time at O(n+m) in space O(p), whereas its worst-case time is O(nm). Since HyperSweep does not focus on particular types of data, optimizing a search algorithm for particular kinds of data would be tough. However, optimizing for scanning and searching the sensitive data stored in specific types of file is still possible. An example would be scanning a PNG file,

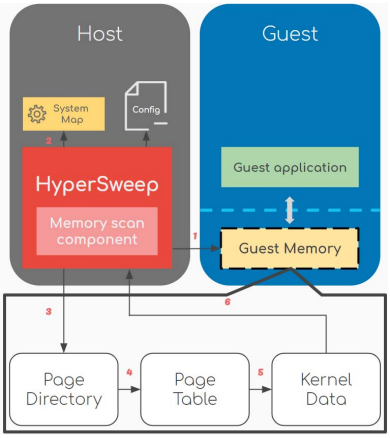

Fig. 3. HyperSweep structural scan

where PNG contains an 8 bytes signature at the beginning of the file. HyperSweep then could be optimized to focus on the comparing the first 8 bytes to differentiate a clean PNG file with a PNG that possibly contains sensitive information. The actual implementation would be more complicated than enhancing the comparison of the 8 bytes signature, but this is an excellent tackle point to improve the scanning efficiency if the administrator wishes explicitly to focus on monitoring PNG files.

IV. Implementation

This section will go through the detail implementation of a HyperSweep prototype. HyperSweep will run on a Linux host machine while operating with regular messenger applications like LINE or Facebook, or web browsers like Firefox or Chrome.

Even though HyperSweep works on both popular hypervisors such as Xen and KVM, yet there are already many monitoring related works or systems in the market that focuses on Xen hypervisor environment because Xen was released in 2003 while KVM initial release was in the late 1990s. Hence, the focus of HyperSweep implementation is more on the KVM hypervisor instead of Xen to compete with the existing project cooperating with Xen and achieve similar if not better results. However, the setup could also be implemented on Xen hypervisor.

A. QEMU-KVM Patch

HyperSweep runs on KVM and Linux x86 architecture machine. The performance of HyperSweep is dreadful if HyperSweep accesses KVM VM with GDB (GNU project debugger) access, which is the native and default access mechanism. Therefore, applying a patch to QEMU-KVM, specifically for KVM hypervisor, to create a new access mechanism is a critical factor to accomplish and to improve the performance. The default and fundamental mechanism are implemented on KVM with the Libvirt framework API, or virsh, to manage and obtain the data from the guest VM. However, KVM Libvirt does not provide the necessary API to map and translate the guest's physical address to host's virtual address. Libvirt is entirely different from the *libxenctrl* used in Xen and hence Libvirt is relatively slower to work with using the default setup.

To improve the guest physical to host virtual address translation, HyperSweep requires QEMU and Libvirt to be compiled from the source to enable QEMU Machine Protocol (QMP) command. The patch will modify the QEMU source code with respect to the correct version to create a UNIX socket for connection. Thus it is able to handle the insertion of QMP commands into the source code. With the newly patched QEMU, HyperSweep can issue QMP commands to access the physical memory by creating the UNIX socket to communicate; so when HyperSweep traces through the guest physical address, it will call the function cpu_physical_memory_map from the QEMU to map the host virtual address. The patch dramatically improves the performance by at least one hundred times faster, depending on how many guest physical to host virtual address commands of HyperSweep has requested. If HyperSweep had to issue a full physical scan and to create a physical dump file of the guest OS, HyperSweep would take at least 15-20 minutes to complete the task before the patch while it will only take around 5 seconds to complete the same task after the patch.

B. Memory Scan Component

The memory scan component of the system is responsible for the detection of sensitive data violation. Each process memory VMAs will be scanned as per the ascending order listed in the process list. In other words, the memory scan component goes through lower address memory to the higher address memory. The pseudo code of the main memory scan component in C language is shown in Algorithm 1 and Algorithm 2.

```
procedure memorySweep
    current_proc <− process_list_head(pointer, os_type)
    while current_proc is not list_head(ostype)
        pid <− vmi_read_value_va(current_proc + pid_offset)
        process_name <− vmi_read_string(current_proc +
      name_offset)
        mm_struct_head <− current_proc + linux_mm
        if vmi_read(mm_struct_head) is not NIL then
            for all mm_struct_field in mm_struct do
                val_mm_struct_field <− vmi_read(mm_struct_head +
       fieldOffset)
            end for
            ptr_vm_area <− mm_struct_head + vmaHead_Offset
            number_vm_area <− 0
            call vmaHandling(number_vm_area)
        end if
        current_prc <− next_proc
    end while
```

Algorithm 1. memorySweep pseudo code

```
procedure vmaHandling(number_vm_area)
  while ptr_vm_area is not NIL
    if number_vm_area > val_map_count then  //exceed assigned
    pages
      return ERROR_vm_area_number
    else
      for all vm_area_field in vm_area do
        val_vm_area_field <− vmi_read(ptr_vm_area +
    fieldOffset)
        sizePage <− val_vm_startAddr − val_vm_endAddr
        end for
      if (val_vm_startAddr to val_vm_endAddr is not in
    whiteList) and (sizePage is not 0) then
        if pageExist is NIL
          skip  //page is empty
        else if isLazyAllocate(val_vm_startAddr, sizePage) =
    True then
          memoryPage <− vmi_read(size_partialMemory)
        else
          memoryPage <− vmi_read(sizePage)
          end if
        result <− Scan_Algorithm(memoryPage,
    sensitiveDatabase)
        if result = True then
          Output memoryPage to file  //memory page violation
        end if
      end if
    end if
    ptr_vm_area <− next_vm_area
  end while
```

Algorithm 2. vmaHandling pseudo code

The procedure memorySweep investigates each process mm_struct page while the procedure vmaHandling handles the VMAs belongs to each of the mm_struct. The required mm_struct field is assigned concerning the offsets value assigned in the configuration files mentioned in the last section. With this mm_struct field, the HyperSweep can access the VMAs and check for any violation of sensitive information. The procedure vmaHandling takes the number_vm_area, the total number of VMAs being assigned, as the reference to check the existing pages. Also, the size of a page is another important attribute. Guest VM would have not yet assigned the actual physical memory space to the process due to lazy allocation. So, the HyperSweep could encounter a case when the page size of the VMA during the scan is smaller than the assigned size since the page is partially allocated or possibly not allocated by the Linux OS at the time of the scan.

V. Evaluation

This section discusses the results and evaluations of HyperSweep. The HyperSweep is installed on Ubuntu 14.04, 64 bits, kernel version 4.4.0-101-generic with Intel i7-4700 CPU @ 3.40 GHz, and 32GB DDR3-1600 RAM. The guest VMs are running on Ubuntu 14.04 32 bits, kernel version 4.2.0-27-generic with a virtual Central Processing Unit (vCPU), 1 GB of memory, and on Windows 7, 32 bits with one vCPU and 1 GB of memory.

A. Memory Scan Component - A Case Study with Chrome Browser

We use the Google Chrome browser as the test subject for HyperSweep to detect the violation of sensitive information. HyperSweep can detect when a user enters any sensitive information into the web browsers web-page or even Google Chrome's web address search bar. The sensitive information on chrome is shown in Figure 4.

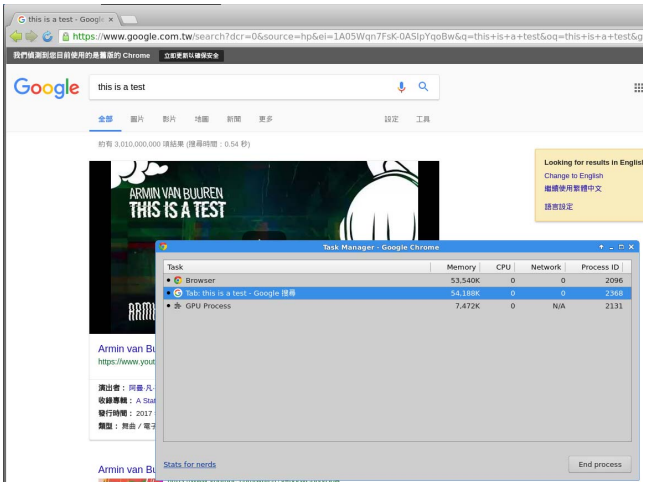

Fig. 4. Sensitive information in Chrome

HyperSweep scans the memory information including the mm_struct and the VMAs to detect sensitive data. Here, the sensitive data is the string "this is a test", and HyperSweep detects a violation and the corresponding log is shown in Figure 5 where the string is found in the 49^{th} VMA.

```
...
−−−start vm_area_struct−−−
Successfully read vm_area_struct pages at cd3806c0!
   vmArea_count: 49
   val_vm_start PID= 0x45004000;
   val_vm_end PID= 0x451fc000;
   val_vm_next PID= 0xcd380780;
   val_vm_prev PID = 0xcd3804e0;
   val_vm_mm PID= 0xcd0fb000;
Begin scanning data section
   vmi_pagetable_lookup_cache: dtb = 220831392, 0
   vm_area_struct page partially scanned, scanned size = 110592
String found at process chrome, VA = 0x45011050, pid = 2368
Did 0 calls in 0.00052 seconds
−−−end vm_area_struct−−−
```

Fig. 5. HyperSweep detects sensitive string in VMA number 49 with interval of (0x45004000 0x451fc000) of Chrome Process, PID = 2368

We continue the test by running the instant messaging application LINE Messenger, which is implemented as a Google Chrome extension. The LINE messaging service has using encryption on all its communication data. Yet HyperSweep is still able to detect the sensitive information by scanning the VMAs used by the LINE messenger. As shown in Figure 6,

the Chrome extension LINE messenger has a PID of 3001 according to the Chrome task manager. The sensitive data chosen here is “this is a test”, and “yet another sensitive data” are found as shown in Figure 7. The corresponding memory dump generated by HyperSweep is shown in Figure 8.

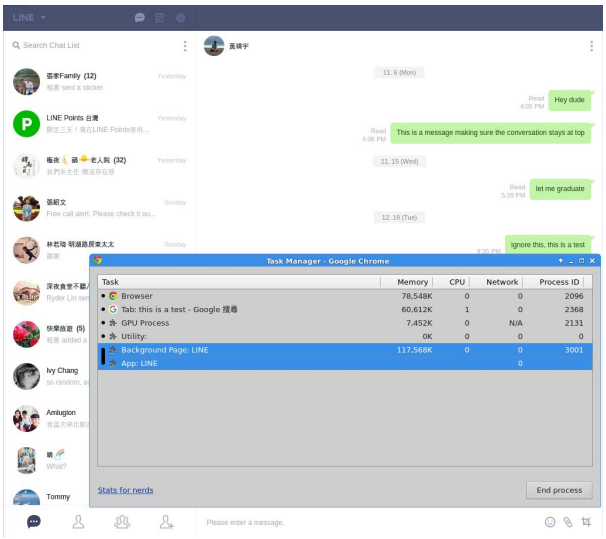

Fig. 6. LINE process PID shown by chrome

```
...
———start vm_area_struct———
Successfully read vm_area_struct pages at f570d780!
    vmArea_count: 44
    val_vm_start PID= 0x2a004000;
    val_vm_end PID= 0x2a1fc000;
    val_vm_next PID= 0xf570d720;
    val_vm_prev PID = 0xf570d660;
    val_vm_mm PID= 0xf2c61000;
Begin scanning data section
    vmi_pagetable_lookup_cache: dtb = 916273152, 0
    vm_area_struct page partially scanned, scanned size = 159744
String found at process chrome, VA = 0x2a020954, pid = 3001
String found at process chrome, VA = 0x2a01f499, pid = 3001
Did 0 calls in 0.001 seconds
———end vm_area_struct———
...
```

Fig. 7. Multiple sensitive strings found in VMA number 44

B. Other Applications and Processes

The summary of the tests on HyperSweep with some popular applications is given in Table I. The test result with the Vim application is showing only partial detection because it stores the text in special data structures so the strings in a sentence could be scattered in the memory. For example, in one of the test run, we noticed that Vim breaks a string into pieces, where each piece is 4 bytes in length.

C. Performance

Overall, the performance of HyperSweep is difficult to measure since the total scanning time depends on the number

```
00 00 00 00 00 00 3A 69 67 6E |.........l..........:ign
20 61 20 74 65 73 74 63 01 00 |ore this, this is a testc..
2F 67 66 2E 6C 69 6E 65 2E 6E |.........2https://gf.line.n
00 00 4F 63 44 33 4D 64 52 47 |aver.jp/enc........OcD3MdRG
44 65 73 63 65 6E 63 2A 02 89 |T10Notice mdRGT10Descenc*..
68 69 73 25 32 30 69 73 25 32 |.re%20this%2C%20this%20is%2
00 00 00 00 32 68 74 74 70 73 |0a%20testc..........2https
6A 70 2F 65 6E 63 01 00 00 00 |://gf.line.naver.jp/enc....
90 2F 00 C8 73 0F 97 20 00 4E |.......H..../..V<./..s.. .N
39 63 63 35 31 65 39 61 61 37 |g.[...2.**...0f119cc51e9aa7
4B 50 40 00 2A 2A 02 36 C0 39 |25a66ecda10472b.`KP@.**.6.9
35 36 35 65 37 61 63 34 39 65 |751dd1375d451f9ae565e7ac49e
C5 65 4B 03 00 00 00 00 DA 65 |3.jp/enc.F`K......eK......e
03 00 00 00 01 00 00 00 18 00 |K.....N.K.....fK..........
00 14 C0 50 00 00 00 00 00 00 |...................P......
DC 33 5F 63 68 61 74 5F 6D 65 |.jp/enc..........3_chat_me
65 6E 63 01 00 00 00 1C 00 00 |ssage_fail_menup/enc.......
69 6E 65 2E 6E 61 76 65 72 2E |....2https://gf.line.naver.
38 64 35 38 30 35 34 63 33 39 |jp/enc*.!.f7814e08d58054c39
6E 63 75 37 63 63 63 61 30 36 |2b54d1568b72.jp/encu7ccca06
62 34 34 63 66 31 65 63 51 5F |e961af717db632399b44cf1ecQ_
38 6E 69 6D 69 60 00 4E A0 52 |OS.l.........+lVq8nimi`.N.R
63 01 00 00 00 1C 00 00 00 00 |..0b-r'Y.R)Rjp/enc.........
65 2E 6E 61 76 65 72 2E 6A 70 |..bhttps://gf.line.naver.jp
79 4C 46 54 20 4D 64 50 42 30 |/enc.........:hcLyLFT MdPB0
```

Fig. 8. Sensitive Information “this is a test” is found

TABLE I
PROCESSES TESTED BY HYPERSWEEP

Process Name	Successful or Fail Scan
Chrome	Successful
Firefox	Successful
LINE (chrome extension)	Successful
Facebook (through browser)	Successful
Terminal (bash)	Successful
Vim (text editor)	Partial
Gedit (native text editor)	Successful
LibreOffice Writer	Successful
LibreOffice Calc (a.k.a. Linuxs Excel)	Successful

of processes running during the scan. Furthermore, the size of each page and the number of VMAs that a process possesses also will have to take into consideration in the performance. The web browser such as Google Chrome requires more memory all the time hence, the sweep time of a single guest VM running the regular software will be used as a generic referencing time to scan a regular guest VM.

A successful scan with a regular page size of 4096 bytes takes time between 1.7×10^{-5} and 3.2×10^{5} seconds. Also, the average time to perform a successful scan per byte is between 4.5×10^{-9} and 5×10^{-9} seconds. The successful scan will go through the information of VMA; then before *memcpy* performs through the page table translation on the 4 KB pages to complete the algorithm it identifies the sensitive information matching. A Swap out or lazy allocation page scan scenario due to a lazy allocated page or swap page with a size of 65536 or 216 bytes runs for approximately 10^{-6} seconds, i.e., a very constant time. The web browser Firefox as the monitoring target is selected for the time to scan a running program. The regular Firefox will have different processes running at the same time. Hence, memory usage of this particular Firefox process has been selected for experimentation. As the actual web content accessed by the Firefox, the performance of one web content process running along the side of Firefox has been tested to see how fast HyperSweep could scan the two processes. The scenario results are shown in the Table II.

For an extensive program like Firefox or Chrome, usually

TABLE II
PAGE AND PROCESS SCAN PERFORMANCE (UBUNTU 14.04 32-BIT GUEST VM)

Scan Type	Performance
Successful Scan per page	190.735 $\sim$ 211.926 MB/Second
Swap out or lazy allocation page scan	1 μs per VMA, regardless of VMA size
Vim (with whitelist)	0.07 $\sim$ 0.09 second (6.45 MB, 74 VMAs)
Firefox (with whitelist)	0.24 $\sim$ 0.3 second (239 MB, 631 VMAs)
Firefox + Web Content (with whitelist)	0.4 $\sim$ 0.45 second (total of 385 MB, 1121 VMAs)

600 or more VMAs required, where many of these VMAs will have a size between 100KB and 200KB with a few above 400KB, depending on the web content that the browser is running. For a smaller process like Vim or bash, it will only have around 50$\sim$70 VMAs which is significantly lower than Chrome or Firefox.

For a runtime scan on a guest Linux VM, a full structural scan will take an average of 1.75$\sim$2.57 seconds to scan all the processes without a whitelist. The complete scanning will take runtime approximately 3.47$\sim$3.62 seconds. On the other hand, guest Windows VM with runtime physical scan will take approximately 3.57$\sim$3.6 seconds to complete the scan. The performance of a comprehensive scan without the QEMU patch has been recorded is very low compared to the patched version. The result in Table III, shows the unpatched vs. patched performance. Note that the whitelist is not enabled in these results, hence with whitelist enable and skipping some known processes, the scanning speed would increase depending on how many memory contents that are skipped.

TABLE III
PERFORMANCE FOR DIFFERENT GUEST VMS

VM OS	Runtime Scan Strategy	Time
CentOS 7	Full Comprehensive Scan without QEMU Patch	20.7 minutes
Ubuntu 14.04 32 bits	Full Structural Scan	1.75 $\sim$ 2.57 seconds
	Full Comprehensive Scan	3.47 $\sim$ 3.62 seconds
	Full Comprehensive Scan without QEMU Patch	19.5 $\sim$ 21.3 minutes
Windows 7 32 bits	Full Comprehensive Scan	3.57 $\sim$ 3.60 seconds

VI. RELATED WORK

A. VMI Applications

Monitoring the hypervisor for malware or data leakage prevention has been researched throughout the years. However, many of them still work only with XEN hypervisor. CloudVisor [11], for example, is developed based on the cloud, which is usually leasing the resources out of all VMs, to run nested virtualization with security monitor in the unmodified XEN-based cloud system. This interposes the interactions of VM memory and its guest for sensitive information protection. The integrity check is by tracking the memory pages of VM while encrypting the contents of the pages when unauthorized mappings are found. The work deals only with the XEN virtualization layer, and it is neither designed to run within KVM nor supports para-virtualization.

Rekall [12] and Volatility [13] provides the memory forensic tools and framework to the hypervisor. Volatility is an old framework which is based on an open collection of tools implemented using Python under the GNU Unix-like OS for extracting digital artifacts from Random Access Memory (RAM) samples. Rekall, on the other hand, is the successor of Volatility. Rekall deals with memory acquisition whereas Volatility does not have this feature. Both Rekall and Volatility runs within the kernel of the target machine. Hence, some kernel-based malware could bypass the memory forensic of the Rekall or Volatility, which leads to the possibility of data leakage or the integrity of the data stored within the target machine.

IntroVirt [14] is a VMI-based Intrusion Detection System (IDS) that mainly focuses on monitoring the malicious activities or policy violation. It does not make changes to the VM Memory, and it supports VM replay functionality with the help of vulnerability-specific predicates. Thus it prevents some response strategies against known but unpatched vulnerability attack. However, IntroVirt incurs performance issue, and it is customized only for XEN hypervisor.

VMwatcher [15] is a VMI-based Malware Detection System (MDS). It is different from LibVMI or IntroVirt because it uses the guest view casting technique to reconstruct semantic-level view of the VM. It also restores the high-fidelity semantic object for the anti-malware software which is existing outside of the target guest VM to create a different perspective. This alternative view can then be compared to discover any inconsistency and to exploit the hidden malware if such discrepancy occurs. Though the guest view casting is very useful, the VMwatcher is challenging to develop due to its technology. Furthermore, VMwatcher is vulnerable to guest view subversion attack and guest caching exploitation.

The Real-Time Deep VMI and its applications (RTKDSM) [16] is implemented with the newer Volatility plugins that run only on XEN hypervisor and it could perform structural scanning on Windows guest VM during runtime. RTKDSM suffers performance penalty due to the induced page faults in the memory.

M. Sharif and W. Cui [17] and Y. Liu et al. [18] have proposed an approach to monitor and secure sensitive information in the memory of an OS. The authors have considered the in-VM approach to detect malware by running the monitoring and security technique within the VM to improve the scanning performance. However, they have not focused on the sensitive information detection. Instead, the authors have focused on detecting and securing the integrity of the system from malware. The risk associated with such design is the higher overhead due to the additional shadow paging technique used. In addition to this, the work relies on hardware to enhance the scanning efficiency and performance and that also increases

the overhead. Also, there is a possibility that kernel-based malware could infiltrate that space since it is implemented in the guest VM. Furthermore, the separate page table created for every process might exhaust the system resources very early since there is a need for significant dynamic allocated memory [17].

B. DLP

Kim et al., [19] have proposed an architecture to detect the SDL from mobile device OS using the Trusted Execution Environment (TEE) Application Program Interface (API) system and ARM's partner, the Global Platform, a separated execution environment in Rich OS. Currently, they only support regular mobile phone services including certificate store, message, camera, gallery except for e-mail service. M. Lu et al., [20] have proposed a work which focuses on enhancing the performance of data leakage prevention, providing a signature database to speed up the comparison of sensitive information. On the other hand, Kongs et al., [21] have introduced the machine learning and information retrieval strategies to identify sensitive content. The authors have looked into the contents that have been summarized, rewritten or the paragraphs which are copy-pasted into a new one by monitoring and categorizing incoming information flow before porting the result into a controlled environment. Thus it ensures better determination of the sensitivity of the document created within the same environment. By recreating the sensitive documents within this controlled environment, they were able to improve detection of incorrectly classified documents that were subject to complex data transformations.

VII. Conclusion

We present a novel hypervisor-based Data Loss Prevention (DLP) system called "HyperSweep". HyperSweep uses Virtual Machine Memory Introspection (VMI) to inspect the memory of a guest VM for sensitive information. Compared with network-based DLP systems, HyperSweep is capable of detecting the presence of sensitive data even when transport encryption is in use. Compared with existing end-point DLP systems, HyperSweep is entirely isolated from the guest system including the guest kernel, making it more robust against potential tampering. The HyperSweep prototype is built on top of the KVM hypervisor. Our experiments indicate that HyperSweep can effectively provide DLP for real-world applications on both Windows and Linux based guest systems. The experiments also indicate that the performance overhead incurred by HyperSweep is moderate.

Acknowledgement

This study is supported in part by the Ministry of Science and Technology of the Republic of China under grant number 104-2221-E-009-104-MY3 and the Taiwan Information Security Center.

References

[1] T. Garfinkel, M. Rosenblum *et al.*, "A Virtual Machine Introspection based Architecture for Intrusion Detection," in *Proceedings of the Network and Distributed System Security Symposium*, vol. 3, 2003, pp. 191–206.

[2] B. Payne, "Simplifying Virtual Machine Introspection Using LibVMI," Sandia National Laboratories, Livermore, California, Tech. Rep., 2012.

[3] A. More and S. Tapaswi, "Virtual Machine Introspection: Towards Bridging the Semantic Gap," *Journal of Cloud Computing*, vol. 3, no. 1, pp. 16–32, Oct 2014.

[4] Y. Hebbal, S. Laniepce, and J.-M. Menaud, "Virtual Machine Introspection: Techniques and Applications," in *Proceedings of the 10th International Conference on Availability, Reliability and Security (ARES)*. Toulouse, France: IEEE, Aug 2015, pp. 676–685.

[5] E. Gessiou, Q. Hieu Vu, and S. Ioannidis, "IRILD: an Information Retrieval based method for Information Leak Detection," in *Proceedings of the 7th European Conference on Computer Network Defense (EC2ND)*. Gothenburg, Sweden: IEEE, Sep 2011, pp. 33–40.

[6] M. Hart, P. Manadhata, and R. Johnson, "Text Classification for Data Loss Prevention," in *Privacy Enhancing Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 18–37.

[7] Data loss prevention market, 2010-2014. [Online]. Available: http://www.radicati.com/wp/wp-content/uploads/2010/12/Data-Loss-Prevention-Market-2010-2014-Brochure.pdf

[8] Libvirt Virtualization API. [Online; accessed 15-January-2018]. [Online]. Available: https://libvirt.org/

[9] QEMU the FAST! processor emulator. [Online; accessed 15-January-2018]. [Online]. Available: https://www.qemu.org/

[10] G. Gonnet and R. Baeza-Yates, "An Analysis of the Karp-Rabin String Matching Algorithm," vol. 34, pp. 271–274, May 1990.

[11] F. Zhang, J. Chen, H. Chen, and B. Zang, "CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2011, pp. 203–216.

[12] Rekall Forensics: Rekall. [Online; accessed 1-January-2018]. [Online]. Available: http://www.rekall-forensic.com/

[13] Volatility Foundation: Volatility. [Online; accessed 1-January-2018]. [Online]. Available: http://www.volatilityfoundation.org

[14] IntroVir: Introspective Virtualization. [Online; accessed 1-January-2018]. [Online]. Available: https://www.ainfosec.com/innovative-products/introvirt/

[15] X. Jiang, X. Wang, and D. Xu, "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*, 2007, pp. 128–138.

[16] J. Hizver and T.-c. Chiueh, "Real-time Deep Virtual Machine Introspection and Its Applications," in *Proceedings of the 10th ACM International Conference on Virtual Execution Environments*, Mar. 2014, pp. 3–14.

[17] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-VM Monitoring Using Hardware Virtualization," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 477–487.

[18] Y. Liu, Y. Xia, H. Guan, B. Zang, and H. Chen, "Concurrent and Consistent Virtual Machine Introspection with Hardware Transactional Memory," in *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture*, Orlando, FL, USA, Jun 2014, pp. 15–27.

[19] G. Kim, J. Lim, and J. Kim, "Mobile Security Solution for Sensitive Data Leakage Prevention," in *Proceedings of the 5th International Conference on Communications and Broadband Networking*. ACM, 2017, pp. 59–64.

[20] M. Lu, P. Chang, J. Li, T. Fan, and W. Zhu, "Data Leakage Prevention for Resource Limited Device," Patent US8 286 253B1, 10 09, 2012. [Online]. Available: https://patents.google.com/patent/US8286253

[21] K. W. Kongsgård, N. A. Nordbotten, F. Mancini, and P. E. Engelstad, "Data Loss Prevention Based on Text Classification in Controlled Environments," in *Proceedings of the International Conference on Information Systems Security*. Springer International Publishing, 2016, pp. 131–150.