Virtual Machine Introspection in a Hybrid Honeypot Architecture

Virtual machine introspection in a hybrid honeypot architecture

Tamas K. Lengyel
University of Connecticut
tamas.lengyel@engr.uconn.edu

Justin Neumann
University of Connecticut
neumann@cse.uconn.edu

Steve Maresca
University of Connecticut
steve.maresca@gmail.com

Bryan D. Payne
Nebula, Inc.
bdpayne@acm.org

Aggelos Kiayias
University of Connecticut
aggelos@cse.uconn.edu

Abstract

With the recent advent of effective and practical virtual machine introspection tools, we revisit the use of hybrid honeypots as a means to implement automated malware collection and analysis. We introduce *VMI-Honeymon*, a high-interaction honeypot monitor which uses virtual machine memory introspection on Xen. *VMI-Honeymon* remains transparent to the monitored virtual machine and bypasses reliance on the untrusted guest kernel by utilizing memory scans for state reconstruction. *VMI-Honeymon* builds on open-source introspection and forensics tools that provide a rich set of information about intrusion and infection processes while enabling the automatic capture of the associated malware binaries. Our experiments show that using *VMI-Honeymon* in a hybrid setup expands the range of malware captures and is effective in capturing both known and unclassified malware samples.

1 Introduction

In recent years there have been significant research and development efforts to move honeypots into an isolated environment provided by virtualization technology. Virtualized environments enable transparent, tamper resistant virtual machine introspection (VMI); however, we must tackle the *semantic gap problem* [4][9]: reconstructing high-level state information from low-level data sources. Many VMI based intrusion detection systems (IDS) and Honeypots aim to bridge the semantic gap by intercepting system calls through the virtual machine monitor (VMM). While this approach has been proven to be an effective method [26][13][6][14], prior works required changes to the hypervisor [6][15][37][35][25] and is vulnerable to in-guest detection of the monitoring environment which can allow malware to disable or modify its behavior [2][27].

In contrast to system call interception, purely memory-based introspection requires no changes to the hypervisor [33][10]. Analysis based upon this technique permits an IDS tool to monitor the virtual machine (VM or guest) without altering its execution, and thus monitoring becomes increasingly transparent to the VM. The major challenge of reconstructing the state of the VM from memory is being able to tackle the semantic gap problem without relying on the guest OS's kernel data-structures. These data-structures could be tampered with by malicious software [1], such as rootkits, and therefore should not be exclusively relied upon [5].

We present a virtualized hybrid honeypot system that uses low-interaction honeypots (LIH) in conjuction with memory introspection based high-interaction honeypots (HIH) to detect and capture malware. To assist in reliably bridging the semantic gap, our system uses forensics tools to bypass the guest kernel to reconstruct the state of the VM. At the same time, inspection from this perspective enhances the transparency of the monitoring environment, as no modification is made to the VMM.

2 Related work

VMI based IDS was first introduced by Garfinkel et. al. [13]. They modified a version of VMware Workstation, a VMM intended for desktop use, to add hooks for observing VM memory, CPU registers, and emulated devices; to reconstruct the state of the VM using the data obtained, they extended a tool more commonly used to examine Linux kernel crash dumps. Similarly, VMwatcher [15] implemented hooks into several VMMs to inspect the memory and filesystem of a guest OS; this data was exported to a separate VM where the memory was compared to a clean state and anti-virus tools run on the filesystem. Both approaches relied heavily upon the guest OS kernel to reconstruct the state of the VM. As a result, they were vulnerable to various kernel subversion techniques: intrusions remained undetectable if they did not modify the filesystem, lacked an anti-virus signature,

or altered kernel data-structures as a means of disguise [1].

Zhang et. al. proposed a purely system call interception based IDS using a modified Xen VMM to analyze and detect intrusions [37]. Similarly, Ether [6], a project developed by Dinaburg et al. is a malware analyzer that uses hardware-assisted virtualization for guests. As Ether utilizes Xen HVM, it executes native CPU instructions, thus does not suffer from incomplete or inaccurate system emulation as other hardware emulators do, such as Qemu. Nevertheless, Pek et. al. showed that Ether is still vulnerable to timing attacks because of implementation issues that reveal the presence of Ether to the guest OS [27].

Argos [28], a Honeypot system developed by Portokalidis et. al. uses system emulation with a modified version of Qemu to perform a hybrid system call interception and memory analysis technique called *taint analysis*, in which bytes originating from the network are flagged ”dirty” and are tracked in the system memory. Argos aims to detect when dirty bytes are being passed for execution to extract the malicious software from memory. While effective, Slowinska et. al. [32] point out that this technique is problematic due to the high ratio of false positives experienced during detection.

Srivastava et. al. implemented a VMI-based firewall [33] called VMwall using the Xen VMM. VMwall captured network flows and, using the XenAccess library [26], correlated them with processes running within a VM. VMwall accomplishes the correlation by extracting information from data-structures of the guest's Linux kernel. Dolan-Gavitt et. al. [10] later noted that the same functionality can be achieved by utilizing forensics tools, such as Volatility, to inspect the guest kernel data-structures in conjunction with XenAccess. As both approaches rely on VM kernel data-structures, they are vulnerable to the same kernel subversion attacks previously mentioned. Our approach is a natural extension to the line of future work proposed by Dolan-Gavitt et. al. and takes advantage of recent developments in the XenAccess successor library to bypass reliance upon untrusted kernel data-structures to reconstruct the state of the VM.

3 Design and Implementation

Our architecture uses open-source tools to integrate virtual machine introspection into a hybrid testbed. In contrast to prior work, our approach minimizes the use of the untrusted guest kernel to improve introspection resiliency against rootkits and other malware. This is achieved by using memory scanning and fingerprinting to accurately reconstruct the state of the VM. In the following section, we provide an in-depth description of the system components, shown in Figure 1.

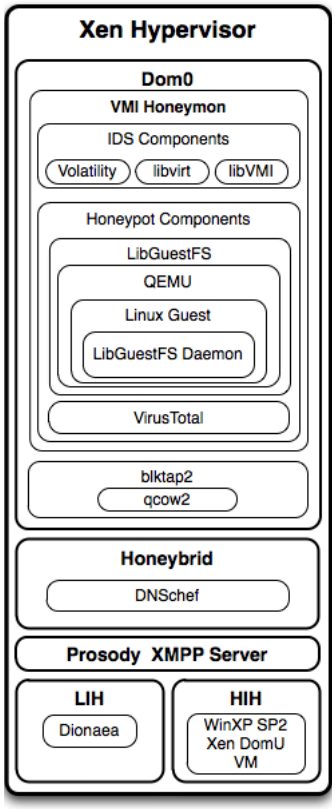

Figure 1: System overview

3.1 Xen, LibVMI and Volatility

During our implementation process we experimented with two open-source hypervisors, Xen [36] and KVM [18]. While our system was operational on both platforms, Xen was chosen for our experiments due to system stability issues experienced with KVM. Xen is a bare-metal hypervisor running at the lowest and most privileged mode of the CPU. In such a system, Xen presides over multiple operating systems, known as domains, with the Xen management domain (dom0) granted privileged access.

LibVMI [24], the successor to the XenAccess library, is an introspection library written in C that builds upon low-level functionality provided by the hypervisor itself. Xen allows visiblilty into many aspects of a virtual machine, including CPU state and guest memory. LibVMI offers a set of functions that encapsulate hypervisor features, providing convenient access to each virtual CPU (including registers), perform page table translations, and read guest memory. Notably, unlike earlier introspection tools, it requires no alteration of the Xen VMM. Running in userspace within dom0, a program

Command	Description	Is scanner?
ssdt	Print the Native and GDI System Service Descriptor Tables.	No
ldrmodules	Cross-reference memory mapped files with the 3 PEB DLL lists.	No
apihooks	Detect IAT, EAT, and Inline hooks in process or kernel memory.	No
idt	Dump the Interrupt Descriptor Table and check for inline API hooks.	No
gdt	Dump the Global Descriptor Table.	No
callbacks	Print kernel callbacks of various types.	No
psscan	Scan memory for EPROCESS objects.	Yes
modscan	Scan memory for LDR_DATA_TABLE_ENTRY objects.	Yes
driverscan	Scan memory for DRIVER_OBJECT objects.	Yes
filescan	Scan memory for FILE_OBJECT objects.	Yes
mutantscan	Scan memory for KMUTANT objects.	Yes
thrdscan	Scan memory for ETHREAD objects.	Yes
sockscan	Scan memory for socket objects.	Yes
svcscan	Scan the Service Control Manager for information on Windows services.	Yes

Table 1: Volatility tests utilized.

utilizing LibVMI is isolated from the monitored guest and therefore protected from tampering by the software inside the VM. As no changes are made to the VMM, monitoring with LibVMI dodges timing attacks that rely upon observation of time skew introduced by additional hooks in the VMM. Similarly, the memory introspection makes no changes to the VM's memory during inspection; therefore, the presence of the monitoring environment remains transparent to the guest.

The LibVMI API functions are accessible via a Python interface which is used by a Volatility extension to access the memory of the guest VM. Volatility is the most popular open source memory forensics tool, incorporating templates for Windows and Linux, and several plugins such as seen in Table 1. While some of the Volatility plugins rely upon the guest kernel entry-points (such as the linked list of running processes) to access datastructures, a set of *Scanner* plugins are also available that bypass the guest kernel entirely to retrieve these structures. This analysis method allows detection of hidden or otherwise disguised kernel data.

The *Scanner* plugins operate by fingerprinting each byte in the inspected memory as a candidate for the target datastructure. As currently LibVMI doesn't provide event-driven monitoring, in our setup the Volatility scans are initiated by timers and by detecting network events.

3.2 Honeybrid, DNSchef and Dionaea

One of our design objectives was to provide a separation between intrusions on the HIH so that the memory footprint we obtain can be assigned to a single intrusion session. Another objective was to tightly control the outgoing connection attempts initiated by HIH to minimize the security risk an infection poses to other entities. Figure 2 shows the layout of our network setup.

Honeybrid [3] is an open-source honeynet coordinator that is designed to reduce the load on HIHs by utilizing low-interaction honeypots (LIH) as filters on the incoming traffic. Honeybrid also provides fine-grained control over outgoing connections from the honeynet, which can be easily extended by creating additional decision modules for Honeybrid.

In our setup, we used the open-source Dionaea honeypot [7] as our LIH, and we created a Honeybrid module to monitor Dionaea's actions via XMPP messages to prevent repeated captures from the same IP. The XMPP messages are exchanged between Dionaea and Honeybrid through a Prosody XMPP server [29]. Utilizing these messages Honeybrid filters out IPs that have previously dropped a payload on the LIH. Another Honeybrid module was created to restrict the use of the HIH to one attacker at a time and to initiate the Volatility scans to detect intrusions when a particular event occurs, such as a connection time-out.

The control module we created for Honeybrid redirects outgoing DNS queries to DNSchef [8] for logging purposes and only allows outgoing connections back to the attacker. All other connection attempts result in Honeybrid initiatiating a final Volatility scan of the VM and reverting the VM to the virgin state. When Honeybrid observes no traffic to the HIH from the attacker for two minutes, the same actions are taken: initiating Volatility scans and reverting the VM.

To avoid a single attacker taking over our HIH for an extended period of time, we placed an additional timer on the attack sessions which sets an absolute allowed timeframe of 10 minutes. After 10 minutes the attacker is redirected to the LIH and the HIH is scanned for intrusions and reverted to the virgin state. We chose 10 minutes as a maximum limit because in our observation with Dionaea infections occured within a minute of the connection being established while others reported an average of three minutes [17]. However, malware may choose to wait longer as a detection avoidance technique.

3.3 The HIH

The HIH was running Windows XP SP2 with 128MB RAM as a VM. The configuration of Windows was slightly modified to simplify tracking of memory

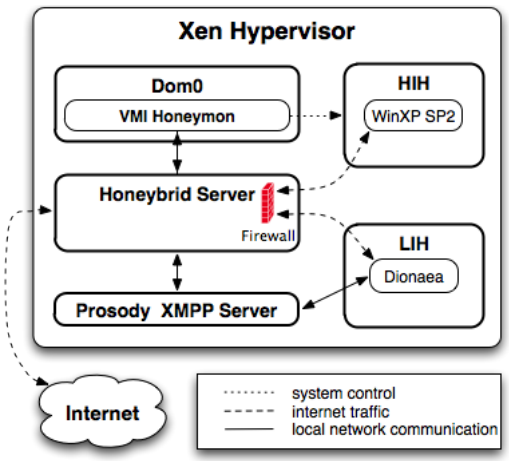

Figure 2: Communication diagram

changes that occur upon intrusion. Automated system processes were disabled such as automatic updates, automatic defragmentation and screensavers. An alternative approach would be to whitelist memory changes associated with these processes. We installed no additional software in the HIH and therefore only the default TCP ports were open: 135, 139 and 445.

Since LibVMI and Volatility supports all versions of Windows, *VMI-Honeymon* could also operate on a wide-range of HIHs. So far we have only experimented with Windows XP SP2 as it had been the standard platform for other introspection based Honeypots [6][9].

3.4 VMI-Honeymon, LibVirt and qcow2

To perform anomaly detection in the output of Volatility's various plugins, we created *VMI-Honeymon* (*honey*pot *mon*itor), written in Perl, to execute Volatility and parse the results in parallel. In our setup, *VMI-Honeymon* resides in a separate domain from Honeybrid, listening to commands through a TCP socket. The anomaly detector compares the Volatility results obtained from the live HIH to results that were obtained from the virgin state of the HIH. By checking and comparing all elements, *VMI-Honeymon* flags all deviations as an anomaly. This method has been effective in detecting a wide range of changes in the HIH during intrusions which is further discussed in Section 4.

To automate back-up and restore procedures, *VMI-Honeymon* takes advantage of functionality provided by LibVirt[20] to automatically save the entire memory of the HIH and revert it to the virgin state when required. By saving the entire memory and filesystem of the VM in an initial "snapshot" operation, *VMI-Honeymon* can quickly revert the HIH to that virgin state without requiring a complete reboot of VM. To stream-line the process of filesystem restoration, we utilize the qcow2 (Qemu copy-on-write) format through the Xen blktap2 driver. Qcow2 works by utilizing a base-image of the filesystem and a copy-on-write image that keeps only the changes that were made. Since no changes are written to the base-image during operations on the qcow2 image, the two main advantages are its small storage requirement and the ability to keep infections in a contained environment.

3.5 Libguestfs and VirusTotal

By utilizing Volatility's *filescan* plugin, *VMI-Honeymon* can quickly detect files that are likely candidates for malware binaries. To extract these binaries, we utilize Libguestfs [19], an open source library developed by RedHat intended for analysis and manipulation of guest filesystems. Libguestfs operates by launching a small Linux appliance inside Qemu and attaches to the filesystem of the running VM. In our setup Libguestfs attaches to the guest filesystem in read-only mode to avoid creating discrepancies in the filesystem.

As *filescan* provides a very rich set of information, some of the files that were flagged by the anomaly detector were not actually changed but have simply been opened by processes. To filter out these false-positives, VMI-Honeymon compares the SHA1 checksum of the candidate files in the running VM to the virgin state and only extracts new files or files with changed checksums.

The extracted files are further checked by *VMI-Honeymon* to determine if the file is a binary to eliminate the extraction of log files which are less likely to contain infections. During the extraction process, *VMI-Honeymon* also submits the samples to VirusTotal (VT) [34] for analysis. VirusTotal is an online resource which provides an API for automatically scanning samples by a set of popular antivirus software. We used VirusTotal to estimate the number of binaries containing malware. As VirusTotal uses 42 antivirus products to scan the submitted samples, we can effectively determine whether the sample is classified or unknown.

4 Tests and Experiments

To evaluate our system's performance and effectiveness, we have conducted several tests and experiments which are discussed in the following section.

4.1 Performance

During regular operation of *VMI-Honeymon*, Volatility scans and baseline comparison took an average of 30 seconds; reverting the VM to a clean state took an average of 25 seconds. We performed further benchmarks of our system with varying VM memory sizes for which

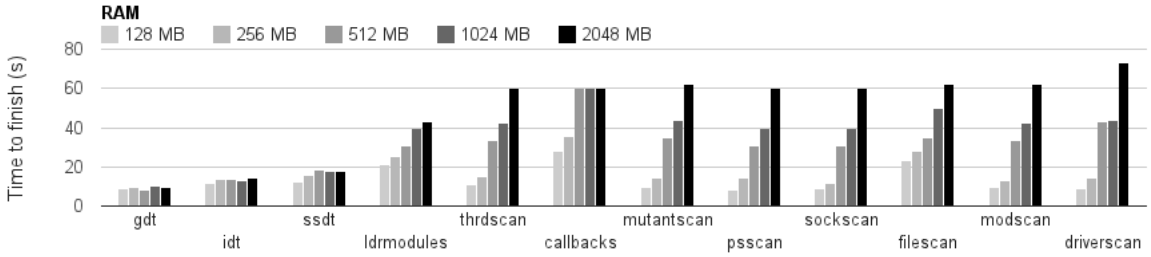

Figure 3: Benchmarks of Volatility scans

the results can be seen in Figure 4. The hardware specs of our system were as follows: second generation Intel i7-2600 quad-core CPU with Intel VT-x and VT-d, Intel DQ67SW motherboard, 16GB DDR3 1333Mhz RAM and two 1.5TB SATA 6.0Gb/s 5900RPM hard-drives using Intel Rapid Storage Technology in RAID 1.

The *snapshot* operation consists of creating a memory backup, a filesystem backup and running initial Volatility scans on the VM. Libguestfs checksum creation of the entire filesystem is a separate step in the snapshot operation not affected by the memory size of the VM and which takes an additional 5-6 minutes to finish (not shown on Figure 4). From our benchmarks it is clear that the memory scanning plugins of Volatility take longer with growing memory size, seen in Figure 3. Nevertheless, even with 2GB of RAM the majority of the scans finished in about a minute.

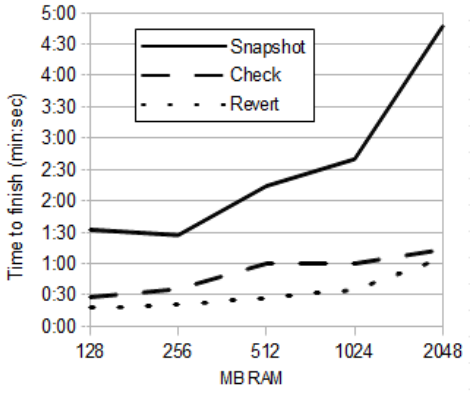

Figure 4: Benchmarks of VMI-Honeymon

4.2 Testing with Metasploit

To verify that our system is able to detect intrusions, we exploited the HIH using Metasploit's *ms08_067_netapi* [31] remote code execution exploit for the SMB stack, an exploit also used by the Conficker malware [22]. This exploit was chosen as Conficker infections had been observed on our LIH with Dionaea on multiple occasions. The anomaly detection did indeed pick up the changes in the memory, particularly the presence of our remote shell, cmd.exe and the presence of a new TCP socket. This intrusion resulted in a total of 127 changes in memory, predominantly new threads spawning according to *thrdscan*, various dll's being loaded into memory according to *ldrmodules* and those dll's being opened for reading according to *filescan*.

We conducted an additional test where we used Metasploit's auxiliary *smb_version* scan to fingerprint the HIH and ran our anomaly detection tool to see if a simple fingerprint operation results in detectable changes. Indeed, running *smb_version* resulted in a set of changes in *filescan* and *ldrmodules* in relation to the kernel modules "spools" and "browser". While the number of changes was small, six changes in the output of *filescan* and a single change in *ldrmodules*, this test illustrates the sensitivity of our anomaly detector.

4.3 Rootkits

To further test our system we obtained recent samples from contagioexchange [23] of the "Sinowal Mebroot Torpig" family and manually infected the HIH with the trojan (md5sum 13ce4cd747e450a129d900e8423-15328). This malware was chosen as it is known for disabling itself when it detects a virtualized environment [21][30]. We observed significant changes all around the memory footprint of the VM, most notably, changes in the Interrupt Descriptor Table (IDT) and the Global Descriptor Table (GDT). A total number of 78 changes were detected, including changes with *psscan*, *mutantscan*, *filescan* and *ldrmodules*. The only filesystem change occurred during execution of the malicious binary we placed in the HIH. No outgoing connection attempts were detected during this infection. Table 2 shows a snippet of the memory changes detected by *VMI-Honeymon*.

Similar results were obtained while infecting the HIH with a TDL4 sample (md5sum 1ca0ca80bf70ca9-99be809edc2606ac0). This malware was also chosen for the known behaviour of disabling itself when a virtualized environment is detected [12]. The infection trig-

gered detection by modifying the IDT and GDT kernel tables and by changing the output of *mutantscan*, *filescan* and *ldrmodules*.

Scan	Result
gdt values	”Sel Base Limit Type DPL Gr Pr”
gdt	”0x38 0x0 0xfff Data RW Ac 3 By P” is missing/changed!
gdt	”0x40 0x400 0xffff Data RW 3 By P” is missing/changed!
gdt	New element: ”0x38 0x7ffdf000 0xfff Data RW Ac 3 By P”
gdt	New element: ”0x40 0x400 0xffff Data RW Ac 3 By P”
idt values	”Index Selector Function Value [Details]”
idt	”21 8 - 0x0” is missing/changed!
idt	New element: ”21 C7 - 0x0”

Table 2: Sinowal Mebroot Torpig detection (snippet).

4.4 Live captures

To expose our system to malicious traffic, it was placed on a University network with the University firewall configured to allow all incoming and outgoing connections. Nevertheless we discovered that traffic originating from outside the University network is still being filtered by the University's ISP which blocks some of the incoming malicious traffic. During our initial Metasploit tests described above, our external IP was blocked from accessing the University network for approximately ten minutes. Nonetheless external malicious traffic still reaches our system, where we recorded 1,158,477 TCP and 467,173 UDP connections in two weeks.

The HIH was exposed to 6,335 connections during this time in 1,980 sessions. *VMI-Honeymon* extracted a total of 886 unique binaries out of which 236 were verified as being malicious by VirusTotal. Dionaea in the same time captured 1,411 binaries out of which 431 were verified by VirusTotal.

One of the worms that has triggered the vast majority of Dionaea captures on our LIH is Conficker. Similarly, we have observed several Conficker infections on our HIH, constituting 14% of the total number of unique captures and 53% of all captures that had a detection with VirusTotal. We observed several variants of Conficker trying to connect to the default gateway on port 445, possibly trying to propagate on the local network (LAN), and also trying to initiate connections to unknown webservers. A detection and capture snippet of Conficker can be seen in Table 3. Another subset of samples, an infection we have not observed previously with Dionaea, triggered detection only with McAfee-GW-Edition according to VirusTotal with a detection named ”Heuristic.BehavesLike.Exploit.CodeExec.L”. Figure 5 shows details of unique binary captures with VirusTotal detection, both from the LIH and HIH.

We further evaluated how many infections tried es-

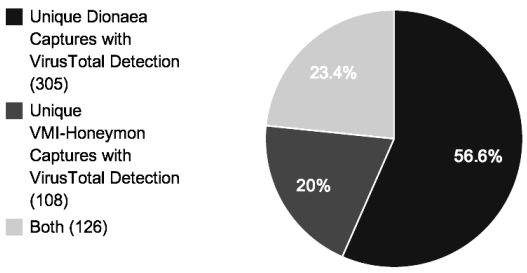

Figure 5: Unique binary captures with VirusTotal detection

tablishing a connection to an IP other than the attacker, which can be seen in Figure 6. The results show us that sessions which resulted in the HIH trying to connect out had a higher number of binary captures with 98% of the captures being verified by VirusTotal. On the other hand, the majority of the sessions with no outgoing connection attempts resulted in no modification to the filesystem. Furthermore, only 26% of the files extracted during these sessions were verified by VirusTotal. As such, it is reasonable to assert that an outgoing connection attempt is a better indicator of a succesfull exploit but it should not be relied upon as the sole indicator.

Action	Result
sockscan values	”Offset PID Port Proto Address Create Time”
sockscan	New element: ”0x00a6e7a0 984 1039 6 TCP 0.0.0.0 2012-04-10 04:10:37”
sockscan	New element: ”0x01085e98 984 1032 17 UDP 127.0.0.1 2012-04-10 04:09:48”
sockscan	New element: ”0x0109ae98 984 7054 6 TCP 0.0.0.0 2012-04-10 04:10:20”
filescan values	”Phys.Addr. Obj Type #Ptr #Hnd Access Name”
filescan	New element: ”0x010c7120 0x80e94ad0 1 0 R–r-d /WINDOWS/system32/fmrmj.dll”
CAPTURE	”/WINDOWS/system32/fmrmj.dll”

Table 3: Conficker infection (snippet).

5 Lessons learned

Operating a HIH has many benefits over a LIH. Foremost, an attacker can quickly discover that the services provided by the LIH are emulated [16] and thus avoid sending a payload. The services provided by the HIH are not emulated and as such, remote fingerprinting in this way will be ineffective. In our setup we utilized a hybrid-approach in which the presence of the LIH is detectable. Furthermore, the presence of Honeybrid is detectable through TCP timestamps when a connection is forwarded to the HIH therefore Honeybrid has to be extended to fabricate timestamps that hide the redirection process from the remote host.

As the HIH runs in fully-virtualized mode, it does not suffer from incomplete or inaccurate system emulation.

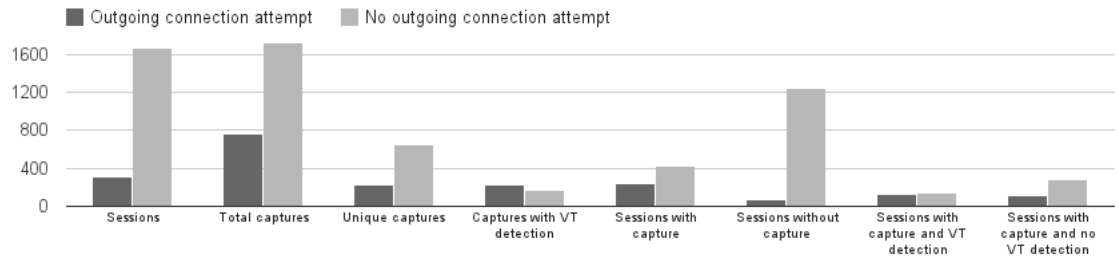

Figure 6: Honeypot statistics. Note: A *Connection Attempt* is defined as any attempt made by the HIH to connect to an IP other than the attacker's IP. A *Session* is defined as all interactions with an attacker.

While it is still possible to discover that the HIH is a VM once the attacker is inside, the extra monitoring layer utilized by *VMI-Honeymon* will remain transparent. Possible timing-attacks to discover our monitoring environment might attempt to measure the memory bandwidth fluctuation caused by our scans. Nevertheless, memory bandwidth fluctuation is not a unique sign of a monitoring environment as running multiple guests can cause the same symptoms.

From the results we obtained from live captures, it is clear that extracting binaries based on filesystem changes can be circumvented or evaded. A malware may choose to remain in memory for a period of time or require some other trigger, such as connecting back to the command and control center, before modifying the filesystem (if at all). In accordance with our containment policy these attempts are not allowed to leave our network, therefore a more refined containment policy might be required. As future work, we are also planning to implement the extraction of the memory space of suspicious processes with Volatility to overcome the limitation of filesystem-based extraction.

While entirely bypassing our anomaly detection would require precise knowledge of the initial state of the HIH, it is still possible that parts of an intrusion could go undetected. Memory polling on a live VM means that memory regions that are in fast-flux might not be properly fingerprinted by Volatility. Additionally, the scanners progress linearly from from the first byte of the inspected memory, therefore it is possible that a malware could move its memory structures to a region that has already been scanned and thus avoid detection. This approach would require the malware to completely erase or overwrite its original memory location and is also dependent upon precise knowledge of when the tests are initiated and how long it takes for each scan to progress through the memory. Both shortcomings could be avoided by pausing the HIH while the Volatity checks are running but this approach would make the presence of *VMI-Honeymon* detectable to the VM by observing the time-dilation caused by the pause. An alternative implementation could use a shadow-copy of the VM memory that remains static for the duration of the scans using copy-on-write techniques as presented by Vrable et. al. [35].

Another limitation of memory scanning is that if the malware alters the data-structures by overwriting non-essential fields used to fingerprint the memory location, it could avoid detection. This limitation could be mitigated by creating robust signitures for the kernel data-structures, outlined in [11], which makes it significantly harder for malware to hide from scanning.

6 Conclusion

The system we presented is a proof-of-concept integration of a virtualized, hybrid honeypot architecture with memory introspection and forensics tools. We were able to capture and analyze 2,297 malware samples in a period of two weeks, of which 71% were unclassified by antivirus vendors at the time of capture. By melding forensics tools with live memory introspection to implement an automated high-interaction honeypot, we were able to expand the range of malware binary captures by 25% compared to operating only a LIH, as shown on Figure 5. While calculating the number of unique binaries captured based on md5 sums is standard, it is worth noting that these numbers are adversely effected by malware packing and do not necessarily mean a unique or new malware. Future work will establish more accurate classification of the samples by clustering the observed memory changes to identify malware families.

While our implementation is an effective and practical solution to achieve automated malware capture, many opportunities remain for future enhancements. The semantic gap problem remains an open research issue, while advancements in introspection libraries continue to narrow the divide. Expanding the scope of our methodology to monitor other operating systems as high-interaction honeypots, while completely supported, also remains unexplored.

References

[1] BAHRAM, S., JIANG, X., WANG, Z., GRACE, M., LI, J., SRINIVASAN, D., RHEE, J., AND XU, D. Dksm: Subverting virtual machine introspection for fun and profit. In *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems* (Washington, DC, USA, 2010), SRDS '10, IEEE Computer Society, pp. 82–91.

[2] BALZAROTTI, D., COVA, M., KARLBERGER, C., KIRDA, E., KRUEGEL, C., AND VIGNA, G. Efficient detection of split personalities in malware. In *NDSS* (2010), The Internet Society.

[3] BERTHIER, R. G. *Advanced honeypot architecture for network threats quantification*. PhD thesis, University of Maryland, College Park, MD, USA, 2009. AAI3359256.

[4] CHEN, P. M., AND NOBLE, B. D. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems* (Washington, DC, USA, 2001), HOTOS '01, IEEE Computer Society, pp. 133–.

[5] CHRISTODORESCU, M., SAILER, R., SCHALES, D. L., SGANDURRA, D., AND ZAMBONI, D. Cloud security is not (just) virtualization security: a short paper. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (New York, NY, USA, 2009), CCSW '09, ACM, pp. 97–102.

[6] DINABURG, A., ROYAL, P., SHARIF, M. I., AND LEE, W. Ether: malware analysis via hardware virtualization extensions. In *ACM Conference on Computer and Communications Security* (2008), P. Ning, P. F. Syverson, and S. Jha, Eds., ACM, pp. 51–62.

[7] DIONAEA. catches bugs. `http://dionaea.carnivore.it`, March 16 2012.

[8] DNSCHEF. `http://thesprawl.org/projects/dnschef`, March 16 2012.

[9] DOLAN-GAVITT, B., LEEK, T., ZHIVICH, M., GIFFIN, J. T., AND LEE, W. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *IEEE Symposium on Security and Privacy* (2011), IEEE Computer Society, pp. 297–312.

[10] DOLAN-GAVITT, B., PAYNE, B., AND LEE, W. Leveraging forensic tools for virtual machine introspection. Gt-cs-11-05, Georgia Institute of Technology, 2011.

[11] DOLAN-GAVITT, B., SRIVASTAVA, A., TRAYNOR, P., AND GIFFIN, J. Robust signatures for kernel data structures. In *Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), CCS '09, ACM, pp. 566–577.

[12] ESET. `http://blog.eset.com/2011/10/18/tdl4-rebooted`, June 9 2012.

[13] GARFINKEL, T., AND ROSENBLUM, M. A virtual machine introspection based architecture for intrusion detection. In *NDSS* (2003), The Internet Society.

[14] HOFMEYR, S. A., SOMAYAJI, A., AND FORREST., S. Intrusion detection using sequences of system calls. *Journal of Computer Security 6* (1998), 151–180.

[15] JIANG, X., WANG, X., AND XU, D. Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction. *ACM Trans. Inf. Syst. Secur 13*, 2 (2010).

[16] KERI, M. Detecting dionaea honeypot using nmap. `http://blog.prowling.nu/2012/04/detecting-dionaea-honeypot-using-nmap.html`, April 3 2012.

[17] KREIBICH, C., WEAVER, N., KANICH, C., CUI, W., AND PAXSON, V. Gq: practical containment for measuring modern malware systems. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (New York, NY, USA, 2011), IMC '11, ACM, pp. 397–412.

[18] KVM. Kernel based virtual machine. `http://www.linux-kvm.org`, April 7 2012.

[19] LIBGUESTFS. Library for accessing and modifying vm disk images. `http://libguestfs.org`, April 10 2012.

[20] LIBVIRT. The virtualization api. `http://libvirt.org`, April 10 2012.

[21] LIGH, M. Torpig vmm/idt signatures. `http://www.mnin.org/write/2006_torpigsigs.pdf`, 2006.

[22] MCAFEE. `http://blogs.mcafee.com/mcafee-labs/conficker-worm-using-metasploit-payload-to-spread`, June 9 2012.

[23] PARKOUR, M. Contagio exchange. `http://contagioexchange.blogspot.com`, April 4 2012.

[24] PAYNE, B. D. `http://google.code.com/p/vmitools`, April 25 2012.

[25] PAYNE, B. D., CARBONE, M., SHARIF, M., AND LEE, W. Lares: An architecture for secure active monitoring using virtualization. *Security and Privacy, IEEE Symposium on 0* (2008), 233–247.

[26] PAYNE, B. D., AND LEE, W. Secure and flexible monitoring of virtual machines. In *ACSAC* (2007), IEEE Computer Society, pp. 385–397.

[27] PÉK, G., BENCSÁTH, B., AND BUTTYÁN, L. nether: in-guest detection of out-of-the-guest malware analyzers. In *Proceedings of the Fourth European Workshop on System Security* (New York, NY, USA, 2011), EUROSEC '11, ACM, pp. 3:1–3:6.

[28] PORTOKALIDIS, G., SLOWINSKA, A., AND BOS, H. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *EuroSys* (2006), Y. Berbers and W. Zwaenepoel, Eds., ACM, pp. 15–27.

[29] PROSODY. `http://prosody.im`, April 10 2012.

[30] QUIST, D. A. Visualizing compiled executables for malware analysis. *Architecture* (2009), 27–32.

[31] RAPID7. Metasploit penetration testing software. `http://www.metasploit.com`, April 12 2012.

[32] SLOWINSKA, A., AND BOS, H. Pointless tainting?: evaluating the practicality of pointer tainting. In *Proceedings of the 4th ACM European conference on Computer systems* (New York, NY, USA, 2009), EuroSys '09, ACM, pp. 61–74.

[33] SRIVASTAVA, A., AND GIFFIN, J. T. Tamper-resistant, application-aware blocking of malicious network connections. In *RAID* (2008), R. Lippmann, E. Kirda, and A. Trachtenberg, Eds., vol. 5230 of *Lecture Notes in Computer Science*, Springer, pp. 39–58.

[34] VIRUSTOTAL. Free online virus, malware and url scanner. `http://virustotal.com`, April 16 2012.

[35] VRABLE, M., MA, J., CHEN, J., MOORE, D., VANDEKIEFT, E., SNOEREN, A. C., VOELKER, G. M., AND SAVAGE, S. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proceedings of the twentieth ACM symposium on Operating systems principles* (New York, NY, USA, 2005), SOSP '05, ACM, pp. 148–162.

[36] XEN. `http://www.xen.org`, April 16 2012.

[37] ZHANG, X., LI, Q., QING, S., AND ZHANG, H. VNIDA: Building an IDS architecture using VMM-based non-intrusive approach. In *WKDD* (2008), IEEE Computer Society, pp. 594–600.