Towards Hybrid Honeynets via Virtual Machine Introspection and Cloning

Tamas K. Lengyel, Justin Neumann, Steve Maresca, Aggelos Kiayias

University of Connecticut, Computer Science & Engineering Department,
Storrs, CT 06269, USA
{tamas.lengyel,justin.neumann,steven.maresca,aggelos.kiayias}@uconn.edu
http://www.cse.uconn.edu/

Abstract. We present a scalable honeynet system built on Xen using virtual machine introspection and cloning techniques to efficiently and effectively detect intrusions and extract associated malware binaries. By melding forensics tools with live memory introspection, the system is resistant to prior in-guest detection techniques of the monitoring environment and to subversion attacks that may try to hide aspects of an intrusion. By utilizing both copy-on-write disks and memory to create multiple identical high-interaction honeypot clones, the system relaxes the linear scaling of hardware requirements typically associated with scaling such setups. By employing a novel routing approach our system eliminates the need for post-cloning network reconfiguration, allowing the clone honeypots to share IP and MAC addresses while providing concurrent and quarantined access to the network. We deployed our system and tested it with live network traffic, demonstrating its effectiveness and scalability.

Keywords: Honeypot, Honeynet, Introspection, Virtual Machine, Network Security, Memory Forensics, Malware Analysis

1 Introduction and Background

In the last decade there have been significant efforts to push high-interaction honeypots (HIHs) to virtualized environments. Virtualized environments provide many benefits for HIHs as they simplify containment and isolation of infections while providing easy and convenient methods for reverting a compromised HIH to a clean state. Furthermore, virtual environments enable real-time monitoring of the execution, disk and memory of the virtual HIHs, providing a direct way to observe infections as they occur and their effects on the compromised systems.

In order for these observations to provide meaningful, high-level state information, one must tackle a semantic-gap problem: given a virtual machine identify the features that are relevant to malware analysis, cf. [3][5]. While primarily system-call interception based approaches have been employed [7][4][8], recent advances in virtual machine introspection (VMI) techniques based purely upon memory observation have been shown to be an effective and practical solution [15]. Memory based VMI can enable a transparent and tamper resistant

view into the state of a HIH, without revealing the presence of the monitoring environment [12].

While memory introspection based honeypot operations have been shown to be effective, the deployment of the technique on a large-scale honeynet setup presents numerous obstacles. Virtual HIHs provide complete systems and application code for an attacker, thus, a scaling issue arises as the hardware requirements increase linearly with the number of HIHs. Furthermore, networking challenges are present when creating quarantined network connectivity to identical HIH clones without internal "in-guest" network reconfiguration. In-guest network reconfiguration would inadvertently lead to changing the initial memory state of the HIH, making comparative analyses of the HIHs more difficult from a pure memory perspective. This issue leads to the "clone-routing" problem, which asks for a way to concurrently route packets to multiple clones that have identical network configurations.

In this paper we present a practical honeynet deployment utilizing a pure virtual machine memory introspection approach. The system takes advantage of recent developments in the open-source Xen hypervisor to effectively tackle the scalability problem with multiple operating systems as honeypots. We also present a novel approach to the "clone-routing" problem using the open-source Honeybrid engine that enables us to deploy clone HIHs simultaneously without requiring in-guest network reconfiguration.

2 Related Work

While many research papers have been published about implementing IDS and Honeypot solutions based on VMI techniques [7][9], the majority of these techniques approach the semantic-gap problem by intercepting system calls through the virtual machine monitor (VMM). These approaches, while effective, are vulnerable to in-guest detection of the monitoring environment by observation of the time-skew introduced by the system call interception [1][14]. In the following we highlight prior work that focuses on the memory scaling and introspection aspects of VMI based IDS and Honeypot solutions.

In 2005, Vrable et. al. implemented a highly scalable honeynet system named Potemkin using the Xen VMM [16]. Potemkin solved the scaling issue associated with running a large number of nearly identical VMs by introducing memory sharing that de-duplicates identical memory pages present across the VMs.While Potemkin was limited to paravirtualized (PV) Linux systems, later works, such as SnowFlock [11], support fully virtualized (HVM) systems as well. As of the latest version of Xen, Potemkin-style VM cloning of fully virtualized systems is now natively supported, enabling the creation of dense honeynet systems using a wide range of operating systems [10].

In 2008, Srivastava et. al. implemented a VMI-based firewall called VMwall using the Xen VMM [15]. VMwall captured network flows in the most privileged domain (Dom0) and correlated them with processes running within a VM by using the XenAccess library. VMwall accomplishes the correlation by extracting

information from data-structures of the guest Linux kernel. In 2011, Dolan-Gavitt et. al. noted that the same functionality can be achieved by utilizing forensics tools, such as Volatility, which in conjunction with XenAccess allows the inspection of guest kernel data-structures [13].

In 2012, Biedermann et. al. presented a dynamic honeypot cloning system using CoW techniques for cloud platforms which enabled incoming attack traffic to be redirected to a stripped-down clone of the Linux VM under attack [2]. The clone honeypot system's memory was monitored to detect new processes spawning in the honeypot. This monitoring worked in concert with parsing of log files inside the clone's filesystem to detect relevant information about ongoing events. While the cloning techniques' performance was comparable to SnowFlock, the VMI techniques employed were susceptible to in-guest subversion attacks because an exploit could unhook its process structure from the kernel. Furthermore, the use of a production VM as the base for a clone honeypot raises security and manageability concerns, as all sensitive information in the original VM has to be found and stripped out during the cloning procedure. While Biedermann et al. explain stripping the contents of the filesystem as part of the cloning procedure, one must also take into account the non-trivial problem of stripping the CoW RAM of sensitive information as well, otherwise the clone honeypot could leak sensitive information.

In 2012, building upon the XenAccess successor library LibVMI , Volatility and LibGuestFS Lengyel et. al. implemented a pure memory observation based Honeypot monitor, VMI-Honeymon, which was capable of automatically detecting and extracting infections from a Windows XP HIH VM [12]. By taking advantage of Volatility's memory scanning approach for state reconstruction, VMI-Honeymon was shown to be increasingly resistant against kernel manipulation techniques used to evade detection in the monitored VM. VMI-Honeymon was deployed in a hybrid honeypot architecture using Honeybrid, which provided improved resiliency against DoS attacks, such as SYN-floods, while at the same time expanded the range of malware captures by utilizing the low-interaction honeypot (LIH) Dionaea as a fall-back system.

3 System Architecture and Design

Our system is a direct extension VMI-Honeymon, focusing on improving the stealth and tamper resistance of the original VMI-Honeymon system and fusing it with Potemkin-style dense VM deployment. The following is an in-depth description of our system.

3.1 Hybrid setup

Drawing from the operational experiences with the original VMI-Honeymon system, we opted to use a similar hybrid setup with Honeybrid and Dionaea as our LIH. The hybrid honeypot setup fuses the benefits of low- and high-interaction honeypots to enable efficient use of available resources. In the hybrid setup all

incoming connections are handled by the LIH first, utilizing HIHs only when needed to bypass the problem of wasting HIH resources on scans or handling a SYN-flood. The hybrid setup also provides a highly configurable firewall to monitor and contain potential intrusions. Figure 1 shows an overview of our setup.

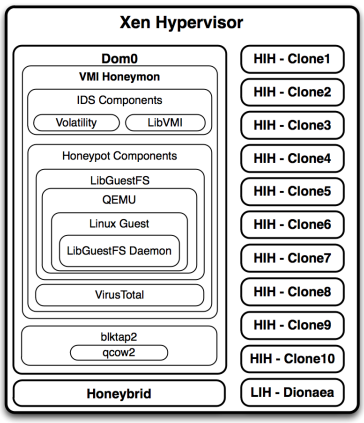

Fig. 1. VMI-Honeymon System Version 2.

While the LIH provides DoS resiliency and emulated services in case the HIH resources are exhausted, the HIHs provide native systems and application code for intrusions. The HIHs are controlled by VMI-Honeymon and are periodically scanned with Volatility or when a network event is detected by Honeybrid. The use of Volatility in conjunction with LibVMI simplifies the state-reconstruction of various HIHs, including Windows XP, Vista and 7 as Volatility provides optimized routines for memory scanning and kernel data-structure fingerprinting for these operating systems.

The Volatility plugins can be characterized by their approach to state reconstruction: scanning or non-scanning plugins. The scanning plugins operate by performing a search for pool tag headers in the VM's memory, while non-scanning plugins operate by following standard kernel data-structures and paths. The trade-off between these two types of plugins is between performance and evasion resistance: scanning the entire memory of a VM takes longer, especially with large memory spaces, but is less likely to miss structures that are unhooked or hidden from the kernel. The Volatility scans in our experiments were limited to running only scanning plugins, which include: open files (filescan), open sockets (sockscan), network connections (connscan), processes (psscan), kernel modules (modscan), drivers (driverscan), mutexes (mutantscan), threads (thrdscan) and registry hives (hivescan).

To avoid introducing heavy memory latency that may lead to side-channel attacks potentially revealing the presence of the monitoring environment by

measuring fluctuations in memory bandwidth, VMI-Honeymon was extended to include a limitation on running parallel memory scans to a pre-defined maximum concurrency level.

3.2 Memory sharing

To achieve dense HIH deployment while avoiding the linear memory requirements associated with running multiple HIHs we take advantage of Xen's native memory sharing subsystem. Memory sharing enable the creation of nearly identical clones that transparently share the memory pages that have not changed during HIH execution. The system is designed so that the origin (parent) VM is paused, and all clones created initially point to the parent's static memory. When a clone writes to memory, Xen performs a copy-on-write (CoW) routine and duplicates the memory page for the clone, providing an optimized use of the overall memory of the physical host.

Recent developments of the memory sharing subsystem enables the creation of nearly identical clones without requiring modifications to Xen itself. While the subsystem is capable of carrying out Potemkin-style flash-cloning of VMs, implementing such cloning remains a future task on the official Xen roadmap. Nevertheless, cloning can also be achieved by performing the standard snapshot-and-restore routine with the XenLight library and de-duplicating the memory pages of the clone afterwards. While the approach is suboptimal, it is sufficient for evaluating the nature of several HIHs in a memory sharing setup.

A key aspect in performing the XenLight (XL) snapshot-restore routine is that the snapshot operation is performed only once when a VM is being designated as a honeypot origin. This snapshot operation also encompasses the scanning of the VM's memory with Volatility and performing a full filesystem fingerprinting with LibGuestFS, so that later infections can be correlated to a known clean-state of the honeypot. By taking advantage of features in the XL utility, the restore routine is further customized, so that the memory snapshot is restored with a dynamically created configuration to place the clone system on a QEMU copy-on-write (qcow2) filesystem.

3.3 Clone-routing

While the combination of CoW RAM and filesystem enables the creation of identical clones, from a networking perspective, the identical clones pose a new challenge: the network interface in each clone will also remain identical, sharing both the MAC and IP address of the original VM. Placing these clones on the same network bridge leads to MAC and IP collisions that prevents proper routing. Both Potemkin and SnowFlock solve the problem of clone-routing by performing post-cloning IP reconfiguration of the VMs. However, this approach is untenable for a memory introspection based system: such reconfiguration will inadvertently change the initial state of the memory in the clone, leading to noisy analysis results when comparing memory states between the clone and its origin. This 'noise' is caused by the process of unpausing the VM to alter

settings, which inadvertently allows suspended processes to resume execution, subsequently causing potentially substantial deviation from the original memory state. It could be argued that unpausing the VM for reconfiguration would alter the state so minimally that it does not appreciably impact the analysis goal. Nevertheless, eliminating any opportunity for the introduction of otherwise avoidable noise appears to be a sensible approach to reach our objectives.

To ensure the creation of truly identical clones and enable pure comparison between a clone and the origin, we retain the MAC and IP of the original VM for each clone. To do so, each clone is placed upon a separate network bridge to provide isolation for the MAC of the clone and avoid a collision. As seen in Figure 2, the clone's bridge is also attached to the VM that runs Honeybrid. This solution enables us to avoid collisions on the bridge, but requires custom routing to be setup on the Honeybrid VM that can identify clones based on the network interface they are attached to instead of their (identical) IP and MAC addresses.

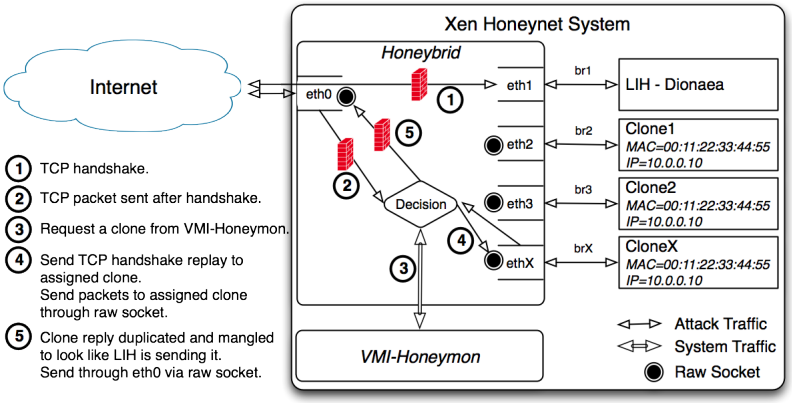

Fig. 2. Clone routing layout - externally initiated.

In order to provide transparent connection switching between the LIH and an HIH, Honeybrid acts as a man-in-the-middle. Using iptables, each incoming connection in the Honeybrid VM is DNATed to the LIH and then queued to be processed by Honeybrid. Each TCP connection performs the TCP handshake with the LIH, and if the connection sends any additional packets, Honeybrid evaluates if the connection should be switched to an HIH. The evaluation is performed in conjunction with VMI-Honeymon where Honeybrid asks for a random available clone from VMI-Honeymon through an SSH tunnel. When there is one available, VMI-Honeymon responds with the clone's name and Honeybrid looks up the clone's interface from the pre-defined configuration file. If VMI-Honeymon reports that all HIHs are taken, the attacker's IP is pinned to use Dionaea. When the connection is switched to an HIH, Honeybrid replays the TCP handshake with the HIH. The incoming packets bound to the LIH there-

after are duplicated and modified to be directed to the clone and transmitted through a raw socket bound to the clone's network interface. The use of raw sockets forces the incoming packets to egress on the proper bridge. Packets from the HIH in return are also duplicated and modified to look like the packet was sent by the LIH.

For connections that are initiated from a clone, which happens for example when an exploit performs a reverse TCP connection, Honeybrid must be aware to route incoming packets for that connection back to the clone that initiated the connection. We use additional routing tables to specify which interface each clone is bound to and by using iptables marks and ip rules we can direct incoming reply packets to specific routing tables which in effect lead to specific clones, shown in Figure 3. We utilize Honeybrid to set the iptables mark on the reply packets by looking up Honeybrid's internal NAT table to identify the original source of the connection. An alternate approach would be to use iptables' CONNMARK save and restore feature to restore connection marks on the packets. We opted to use Honeybrid for this task as it allows for a potential configuration where internally initiated connections are dynamically switched between honeypots as well.

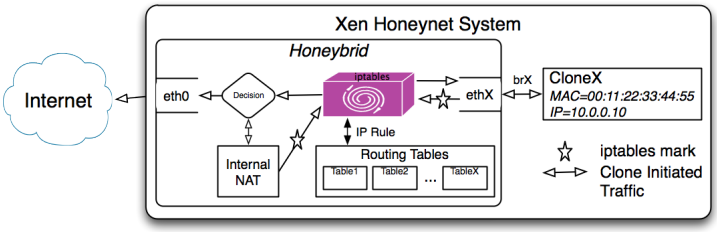

Fig. 3. Clone routing layout - internally initiated.

Since currently the network setup requires manual configuration of the routing tables, network interfaces, iptables marks and ip rules, we used a pre-defined pool of clones for testing. This choice was made for our initial testing purposes and it should be noted that by using Xen's network hotplug features it would be possible to add new clones to the honeynet on-the-fly.

4 Operational experiences

We have conducted several tests and experiments which are discussed in the following section. The tests were focused on the scalability of the memory sharing subsystem when used with Windows XP SP2 x86, Windows XP SP3 x86, and Windows 7 SP1 x86 clones. The experiments were conducted on a single server with the following hardware specs: second generation Intel i7-2600 quad-core CPU, Intel DQ67SW motherboard and 16GB DDR3 1333Mhz RAM. In our tests

the Windows systems were running with the minimum recommended memory, which is 128MB RAM for Windows XP x86, and 1GB RAM for Windows 7 SP1 x86.

4.1 Idle clones

An important aspect of our intrusion detection approach and of effective memory sharing is to limit the memory changes that are not related to an incoming attack. While in Windows XP the number of background processes that generate unrelated memory changes are limited to a handful of services (automatic updates, NTP, background disk defragmentation and background auto-layout), in Windows 7 the number of such services have increased significantly. The effect of these background services on Windows 7 is significant as even within two minutes the amount of shared memory decreases below 25%, effectively requiring over 750MB RAM to be allocated to the clone. At the same time, the clone itself reported only using 26% of its available memory, therefore the allocated CoW memory pages had only short-lived purposes.

By disabling background services which required the allocation of unnecessary resources and polluted our Volatility scans, we were able the minimize the resources allocated to idle clones. The disabled Windows 7 services include prefetch, superfetch, BITS, RAC, indexing, offline files and font cache. Figure 4 shows the resulting memory sharing state of the clones when idle, in terms of shared memory and Figure 5 in terms of additional RAM allocated to the Windows 7 SP1 clones. It is important to note that disabling too many services in the HIH will inevitably impact the attack surface of the HIH, as these services may contain vulnerabilities that could be looked for and/or exploited by the attacker. Since the services we disabled were not listening for incoming connections on the network we deemed their absence to be a reasonable trade-off from a network intrusion perspective. Further examining this performance / detection trade-off is an interesting question that can be tackled in future work.

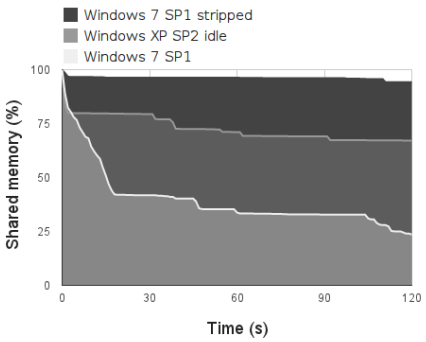

Fig. 4. Clone shared memory when system is idle.

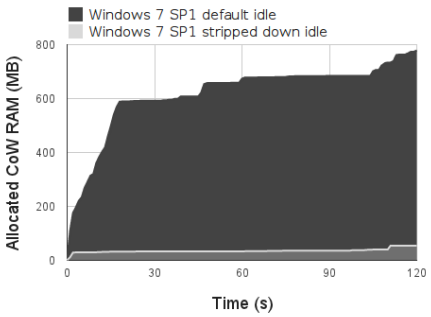

Fig. 5. CoW RAM allocated when system is idle.

4.2 SMB and RDP

The second set of tests we ran on our system were targeting open services in our clones, namely the standard SMB and Remote Desktop services as both of these services have known vulnerabilities. The RDP sessions had significant impact on the RAM allocated to both the Windows XP and Windows 7 clones, reducing the amount of shared memory to 25% in case of the Windows XP clone and 50% for the Windows 7 clone, as seen in Figure 6. In terms of actual RAM allocation, the Windows 7 clone's 50% memory allocation translates to allocating 500MB RAM for the clone, while the Windows XP clone at 75% required 96MB RAM.

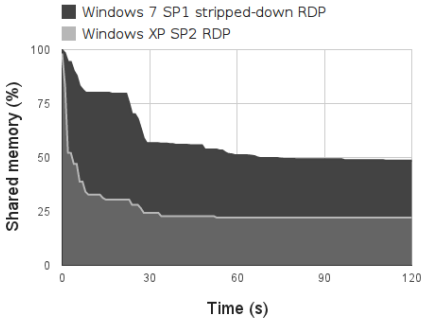

Fig. 6. Clone shared memory after RDP connection.

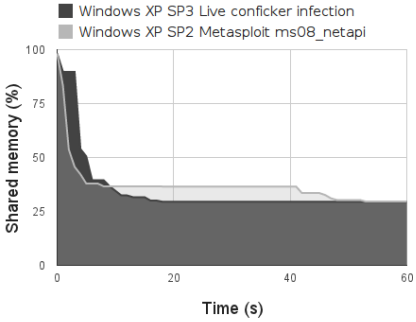

Fig. 7. Clone shared memory after SMB exploitation.

The SMB tests were only conducted on Windows XP clones as Windows 7 SP1's SMB stack has no publicly available exploit. We used a manual Metasploit exploit session (ms08_067_netapi) to benchmark the effect on the Windows XP clone when used with a meterpreter payload that performs a reverse TCP call-back. This exploit was chosen because Conficker uses the same vulnerabilities, which has been observed many times during our live tests. Figure 7 shows the result of the benchmark compared to a live Conficker infection. Only the first 60 seconds were benchmarked since the Conficker infection performed a connection attempt to a third party at that point, triggering our pause-scan-revert operation with VMI-Honeymon. The Windows XP clones retained 25% of their memory in a shared state, which translates to saving 32MB of RAM.

4.3 Live sessions

Our experiments were conducted using multiple HIH back-ends drawn from a pool of clones consisting of five Windows XP SP3 x86 and five Windows 7 SP1 x86 VMs. Each Windows VM was configured with the firewall, automatic updates, time synchronization and memory paging turned off and remote desktop enabled. Windows 7 had additional adjustments as described previously in Section 4.1.

For the live captures we utilized a single IP on a university network with all firewall ports open. Over two weeks of activity, we recorded a total of 52761 connections out of which 6207 were forwarded to an HIH. Currently we forward any incoming connection that passes the TCP handshake to an HIH (if one is available), regardless of whether the HIH is actually listening on the port the attack is targeting. In this way, 1466 forwarded connections never actually established real communication with the HIHs, because these connections targeted ports that were closed (MsSQL, MySQL, SSH, HTTP, VNC).

For the live sessions, one aspect we were interested in was the concurrency of active clones and the amount of memory savings achieved due to CoW RAM. Figure 8 and Figure 9 shows the breakdown of the concurrency that occurred in our system. Figure 10 and 11 show the distribution of the memory remaining shared at the end of the clones' life cycle.

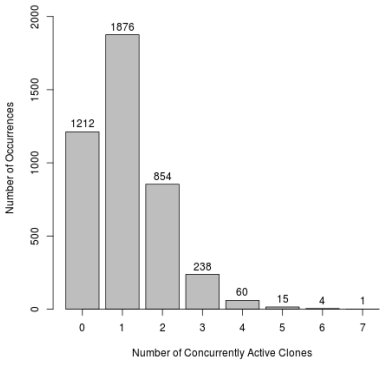

Fig. 8. Clone activity by number of occurrences.

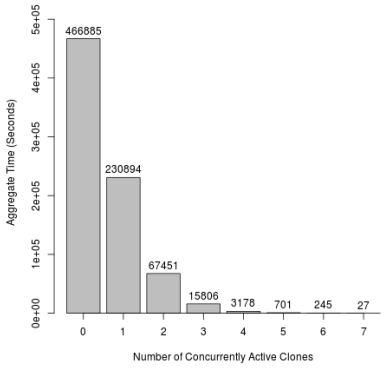

Fig. 9. Clone activity by time spent in each state.

While VMI-Honeymon is configurable to scan the clones periodically during their life-span, we decided to limit such scans to a single instance which happens when the clone reaches its maximum allowed life-span or when a network event is detected. The maximum life-span was set at two minutes, which is cut short if the clone initiates a connection to an IP other than the attacker's. The highest concurrency of active clones was observed as seven, therefore our pool of ten clones was never depleted. The HIHs were actively handling incoming attack traffic 41% of the time during our experiment.

By using the information gathered during these sessions we calculate the projected memory savings when running multiple clones concurrently, shown in Figure 12 and Figure 13. From these projections it is clear that the savings are more significant when the base memory of the HIH is large, as in the case of Windows 7 SP1, allowing for a larger percentage of the overall memory to

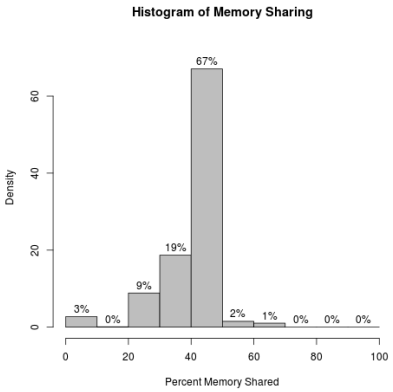

Fig. 10. Shared memory distribution of Windows XP SP3.

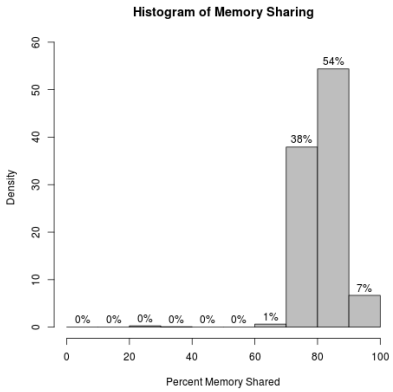

Fig. 11. Shared memory distribution of Windows 7 SP1.

remain shared. We estimate that we would be able to run 40 Windows 7 SP1 clones concurrently and not run out of memory even if all forty clones were three standard deviations above the observed average memory allocation with our 16GB RAM limitation (this would still use only 13.34GB RAM out of the available 16GB). Similarly, we would be able to run 140 Windows XP SP3 clones concurrently and not run out of memory which would allocate 14.6GB RAM assuming all clones are three standard deviations above the observed average.

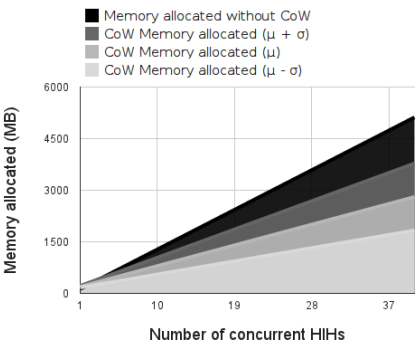

Fig. 12. Projected memory savings of Windows XP SP3. μ=75.52MB σ=10.1MB

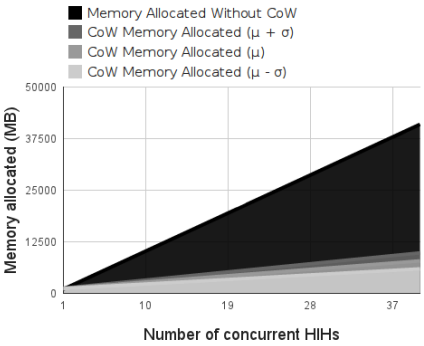

Fig. 13. Projected memory savings of Windows 7 SP1. μ=170.94MB σ=48.3MB

The malware samples we obtained were all Conficker variants verified by VirusTotal and all of the samples were extracted from the Windows XP HIHs. Nevertheless, we have observed several intrusions in our Windows 7 HIHs as

well, which resulted in the clones trying to perform DNS queries. The service exploited during these attack sessions were against the SMB server running on port 445. To allow these intrusions to further interact with the HIH to potentially drop a payload we will be refining our firewall policy to allow some DNS queries and to allow connections to the IP's mapped in the DNS response (with certain rate-limiting applied as to avoid potential malware propagation from within the honeynet).

5 Future Work

Attackers wishing to DoS our system could produce slightly-interactive SYN-flood like traffic which would step just past the initial handshake, causing an HIH to be deployed for every connection, depleting our pre-defined pool of available HIHs. Another DoS approach could use the compromised HIHs to make small changes throughout the memory of the VM in order to produce changes in as many memory pages as possible, making Xen CoW RAM less effective and diminishing the benefits of the memory deduplication. Both of these issues require the system to be modified so that safety checks are included that mitigate these problems. The slightly interactive connection problem could be avoided by dynamically shortening the life-span of the clone when network inactivity is detected instead of running it for a pre-defined period of time. The memory dedup attack could be potentially mitigated by dynamically checking the concurrency of the clones running in the system and their memory sharing stage to automatically pause and halt clones whose memory sharing is approaching a critical state.

While our cloning routine is effective and required no changes to Xen, it can clearly be improved. Further work is necessary to be able to perform flash-cloning rather than restoring the entire memory image to the clone just to be discarded by the memory sharing. Similarly, instead of running a pre-defined pool of clones, it would be beneficial to allow the pool to dynamically balloon up and down to match the incoming attack rate. This inevitably requires Honeybrid to be able to automatically add and remove clones from its routing engine, which could be achieved by careful cordination with VMI-Honeymon and by utilizing Xen's network hotplug features.

Although Volatility's memory scanning and fingerprinting routines are already optimized, there is more room for improvement. Currently each Volatility scan has to evaluate the target memory from start to end independently of one-another. Combining the scans in such a way that Volatility detects all structures by traversing the memory only once would improve performance considerably. However, it should be noted that the pool tag headers that these scans rely on can still be manipulated by rootkits and therefore a more robust scanning approach should be taken into consideration that uses, for example, data-structure invariants for fingerprinting [6].

Building upon the nature of Xen's CoW RAM it would be possible to direct Volatility to examine only the specific memory regions which have changed dur-

ing the execution of the HIH, drastically reducing the memory space the scans have to evaluate. Furthermore, through Xen's memory events subsystem, these directed and combined Volatility scans can potentially allow for real-time monitoring of the HIHs, instead of the current one-time evaluation at the end of the clone's life cycle.

Our current experiments were performed using a single IP on a university network but, as we have shown, the system is capable of effective scaling and it is possible to utilize a darknet to increase the rate of the incoming connections. Furthermore, our experiments focused on x86 versions of Windows XP and Windows 7, but as Volatility supports both 64-bit versions of these operations systems, as well as Linux, it is possible to use these OSs as HIHs as well in future experiments.

6 Conclusion

We have shown a practical solution to deploying a scalable honeynet system on Xen using virtual machine introspection techniques to detect intrusions and extract associated malware binaries. By melding forensics tools with live memory introspection the system remains effectively transparent to in-guest detection of the monitoring environment and is increasingly resilient against in-guest subversion attacks. By utilizing both copy-on-write disks and memory, the system mitigates the linear increase in hardware requirements typically associated with running multiple virtual HIH. Additionally, our novel routing approach eliminates the need for post-cloning network reconfiguration of the HIHs. While our implementation is an effective and practical solution to achieve scalable and automated malware capture, both opportunities and challenges remain for future enhancements.

References

1. Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Srinivasan, D., Rhee, J., Xu, D.: Dksm: Subverting virtual machine introspection for fun and profit. In: Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems. pp. 82–91. SRDS '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/SRDS.2010.39
2. Biedermann, S., Mink, M., Katzenbeisser, S.: Fast dynamic extracted honeypots in cloud computing. In: Proceedings of the 2012 ACM Workshop on Cloud computing security workshop. pp. 13–18. CCSW '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2381913.2381916
3. Chen, P.M., Noble, B.D.: When virtual is better than real. In: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems. pp. 133–. HOTOS '01, IEEE Computer Society, Washington, DC, USA (2001), http://dl.acm.org/citation.cfm?id=874075.876409
4. Dinaburg, A., Royal, P., Sharif, M.I., Lee, W.: Ether: malware analysis via hardware virtualization extensions. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conference on Computer and Communications Security. pp. 51–62. ACM (2008), http://doi.acm.org/10.1145/1455770.1455779

5. Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J.T., Lee, W.: Virtuoso: Narrowing the semantic gap in virtual machine introspection. In: IEEE Symposium on Security and Privacy. pp. 297–312. IEEE Computer Society (2011), http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5955408
6. Dolan-Gavitt, B., Srivastava, A., Traynor, P., Giffin, J.: Robust signatures for kernel data structures. In: Proceedings of the 16th ACM conference on Computer and communications security. pp. 566–577. CCS '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1653662.1653730
7. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: NDSS. The Internet Society (2003), http://www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/13.pdf
8. Hofmeyr, S.A., Somayaji, A., Forrest., S.: Intrusion detection using sequences of system calls. Journal of Computer Security 6, 151–180 (1998)
9. Jiang, X., Wang, X., 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), http://doi.acm.org/10.1145/1698750.1698752
10. Lagar-Cavilla, H.A.: Xen-devel: Cloning a vm and copy-on-write deduplicating memory using cow page sharing in xen 4+. http://lists.xen.org/archives/html/xen-devel/2012-02/msg00259.html (February 2 2012)
11. Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., de Lara, E., Brudno, M., Satyanarayanan, M.: Snowflock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European conference on Computer systems. pp. 1–12. EuroSys '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1519065.1519067
12. Lengyel, T.K., Neumann, J., Maresca, S., Payne, B.D., Kiayias, A.: Virtual machine introspection in a hybrid honeypot architecture. In: Proceedings of the 5th USENIX conference on Cyber Security Experimentation and Test. pp. 5–5. CSET'12, USENIX Association, Berkeley, CA, USA (2012), http://dl.acm.org/citation.cfm?id=2372336.2372343
13. Payne, B.D., Lee, W.: Secure and flexible monitoring of virtual machines. In: ACSAC. pp. 385–397. IEEE Computer Society (2007), http://doi.ieeecomputersociety.org/10.1109/ACSAC.2007.38
14. Pék, G., Bencsáth, B., Buttyán, L.: nether: in-guest detection of out-of-the-guest malware analyzers. In: Proceedings of the Fourth European Workshop on System Security. pp. 3:1–3:6. EUROSEC '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1972551.1972554
15. Srivastava, A., Giffin, J.T.: Tamper-resistant, application-aware blocking of malicious network connections. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID. Lecture Notes in Computer Science, vol. 5230, pp. 39–58. Springer (2008), http://dx.doi.org/10.1007/978-3-540-87403-4_3
16. Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A.C., Voelker, G.M., Savage, S.: Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In: Proceedings of the twentieth ACM symposium on Operating systems principles. pp. 148–162. SOSP '05, ACM, New York, NY, USA (2005), http://doi.acm.org/10.1145/1095810.1095825