DACSA: A Decoupled Architecture for Cloud Security Analysis

Jason Gionta[1], Ahmed Azab[3], William Enck[1], Peng Ning[1], and Xiaolan Zhang[2]

[1]North Carolina State University
{jjgionta,whenck,pning}@ncsu.edu
[2]Google Inc.
{czhang.us}@gmail.com
[3]Samsung Electronics Co., Ltd.
{ahmedmoneeb}@gmail.com

Abstract

Monitoring virtual machine execution from the hypervisor provides new opportunities for evaluating cloud security. Unfortunately, traditional hypervisor based monitoring techniques tightly couple monitoring with internal VM operations and as a result 1) impose unacceptably high overhead to both guest and host environments and 2) do not scale. Towards addressing this problem, we present DACSA, a decoupled "Out-of-VM" cloud analysis architecture for cyber testing. DACSA leverages guest VMs that act as sensors to capture security centric information for analysis. Guest VMs and host environments incur minimal impact. We measure DACSA's impact to VMs at 0-6% and host impact at 0-3% which is only incurred during state acquisition. As a result, DACSA can enable production environments as a testbed for security analysis.

1 Introduction

Security is a top concern for organizations looking to adopt cloud services [3]. Not only do organizations give up control over the infrastructure layer, they also expect to be co-located with VMs that may be compromised. Despite these new security challenges, cloud computing also offers new opportunities for enhancing and evaluating security. In particular, cloud infrastructure can be leveraged as a testbed for enabling and performing security analysis. The large scale of cloud infrastructure allows modern data mining techniques to be applied to massive monitoring datasets to detect patterns of attacks that would otherwise be difficult to glean within a smaller scale environment.

In a cloud infrastructure testbed, client VMs act as sensors for capturing data and identifying malicious activity. For example, port analysis can identify communication endpoints for botnets or code injection in compromised processes. Capturing critical security information regarding client VMs can be enabled using hypervisor based (i.e., Out-of-VM) monitoring of operations [1, 10–12].

Traditional Out-of-VM analysis interposes a layer between the guest OS and the hypervisor which is used to intercept "live" events as they occur in the guest OS [10–12]. For example, Revirt traps and logs all system calls from user to kernel space recording inputs. As such, the monitoring becomes part of the critical path of execution for guest applications, which imposes unacceptably high overheads on host and guest workloads.

In this paper we propose DACSA, an Out-of-VM cloud analysis architecture that decouples the analysis from the capturing of the running memory states of the guests. This decoupling is a conscious design decision that sacrifices "live" monitoring for scalability and performance. We optimize the capturing of memory states to minimize impact on VM operating environments. Then analysis occurs *offline* on the captured state, again minimizing impact on guest and host environments. As a result, client VMs are leveraged as low impact sensors allowing cloud infrastructure to be leveraged as a testbed for security testing.

DACSA is transparent to guest operations by operating Out-of-VM and with minimal impact. We leverage techniques for quickly acquiring the running state of a VM environment with minimal impact to VM execution. We then use memory forensic techniques to extract security relevant features such as process memory or library dependencies. Relevant information can then be used as triggers or analyzed in an automated fashion.

We provide the following contributions:

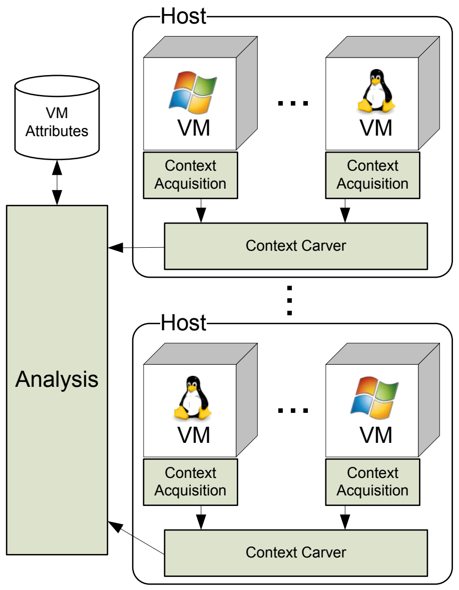

Figure 1: DACSA Architecture: A Decoupled Approach for VM Memory Content Acquisition and Analysis

- We design a generic architecture towards enabling cloud infrastructure as security testbed using VMs as low impact sensors.
- We develop a technique to acquire the running state of a VM environment with minimal impact to the client operating environment.
- We evaluate our architecture and show minimal impact to both the guest and host environments with fixed overhead.

2 Design

The DACSA Architecture consists of three main components: client runtime acquisition, state carving, and analysis. The client runtime acquisition is responsible for extracting the running state of the VM environment. The running state includes the live memory and CPU context at the time the extraction request is made. State carving describes the process of extracting relevant information from the captured running state. Finally, the analysis tool represents a corpus of third party security tools for identifying threats. Figure 1 depicts the DACSA platform. For our design discussion, we use the virtualization software QEMU/KVM to host VMs.

2.1 Client Runtime Acquisition

Testbed analysis requires periodically acquiring the running state of VMs. The process of acquiring the running state of an executing environment is not a new idea. Runtime state acquisition is currently supported by many virtualization platforms [4,5] and in-host memory acquisition tools [6]. However, existing acquisition techniques often suffer from limitations that require either 1) pausing execution for several minutes while state is acquired or 2) processing state while execution continues which leads to unreliable information [6].

Fast Out-of-VM Context Snapshotting

The intuition of fast snapshotting is to perform the minimal amount of work to acquire the working context of the machine. We observe that the majority of work associated with context acquisition techniques [4,5] results from the process of copying data byte-by-byte. If the copy process can be done in an offline manner, machine execution can continue while the context is acquired.

We facilitate copying context in an offline manner by creating a logical copy of machine memory. Logical copies can be made without copying data byte-by-byte and only require creating data structures. As a result, logical copy creation requires minimal time, CPU, and memory resources. Also, the logical copy will remain unchanged while execution continues.

Logical Memory Copy: We use Copy-On-Write (COW) to facilitate creating a logical copy of memory. COW allows multiple entities to share the same set of data without making duplicates. The operating system only needs to manage internal data structures representing the copy.

We create a logical copy of the VM using fork. Fork creates an identical logical copy of the running QEMU process using the Copy-On-Write feature of the host operating system. Once forked, we allow the parent process to continue to run the VM. The child process contains the identical logical copy of the parent which is used to drive analysis.

The above approach discusses a type II hypervisor. Fast snapshots can also be accomplished on a type I hypervisor by setting all guest pages to read-only and handling copying pages on writes. This approach is present by Srivastava et al. towards trusted VM snapshots [13].

Ensuring a Reliable Copy: We stop VM execution prior to the copy creation and flush all asynchronous I/O requests. This is required as allowing two VMs to run using the same resources will lead to a non-deterministic state which will 1) affect the running guest environment and 2) affect correctness of the snapshot. The period the VM is stopped is the time it takes to create the logical copy i.e. the time to fork the QEMU process. Once the process is forked the parent restarts the VM. The child process remains stopped to no longer interfere with execution.

2.2 Context Carving

Carving is a computer forensics term to extract information from raw memory or machine context using foreknowledge about the operating environment. Many mature forensic tools exist to carve out intricate information such as process memory, API hooks, open ports, registry key, etc. Using existing forensic tools, we can immediately carve relevant VM information from the logical copy to be used for analysis.

DACSA uses the logical copy as the datasource for carving. The logical copy can be used to export the VM's physical memory in to a Raw/DD style format which is a common format supported by existing forensic tools [2]. QEMU provides interfaces for reading guest memory by physical address. Memory is written out in a linear fashion from low to high (e.g. Raw/DD). The CPU context is also recorded for assistance in carving.

Unfortunately, dumping memory in Raw/DD style requires writing the entire memory contents of the VM physical memory to disk proceeded by reading the file by forensic tools. As a result, this can have a severe impact on I/O performance for the guest and host. As an alternative, memory forensic techniques can also be adopted to work directly with the logical copy by 1) creating custom carving tool for the specific operating system or 2) providing an interface for forensic tools to work with the logical copy.

Custom carving uses a highly tailored library to extract specific VM information based on the known internal structure of the VM operating system. The extracted information is highly dependent on the analysis being performed.

Interfacing the logical copy directly with forensic tools provides the most benefit without imposing the costs of writing and reading the entire memory contents to/from disk. To build an interface, a shared library is created which hooks I/O operations for the forensic tool. I/O operations that are requested for the logical copy are simulated. For example, open requests for the logical copy will return a fake file descriptor that is tracked. Requests not specific to the logical copy are passed through as normal.

2.3 Analysis

DACSA's analysis component takes information from carving as input for analysis. Information is then sent to a separate machine for analysis. This prevents resource contention and security that could affect both the host and guest operating environments.

Analysis can be done on a per VM basis such as scanning for malware or across VM by clustering features such as library dependencies. Bianchi et al. have previously clustered kernel features can be used to identify unknown rootkit installations [6].

3 Implementation

We prototype the DACSA platform using QEMU/KVM [5] virtualization software. We extend QEMU's monitor interface to allow for new context acquisition and carving requests. Our host is Ubuntu 12.04 64-bit and VM environment is Windows 7 SP1 64-bit.

Our custom carving tool uses the internal structure of Windows to walk the processes in memory. We use the kernel's GS (64-bit)/FS (32bit) register which contains the Kernel Process Control Region (KPCR) to find the head EPROCESS data structure. In some cases, the CPU context may contain userspace data and thus the GS/FS register points to the Thread Environment Block (TEB). In this case we can use a cached KPCR value or linear scan the physical memory to find the KPCR data structure [2].

Our I/O interception library is a generic shared library for use with *nix based operating systems which support dynamic linking. The shared library contains functions to handle basic I/O operations such as open, close, read, lseek, fstat, etc. Upon an open, lstat, or xstat call, our interception library checks the name of the file against a pre-known value representing the logical memory copy and takes appropriate action to handle the operation. Shared memory is used to enable communication and data marshaling between the logical copy and I/O interception library. Read operations result in data being copied from the logical copy to the provided buffer.

We use the above carving mechanisms to extract all process virtual memory. In Windows, this can be done by walking the VAD (Virtual Address Descriptor) tree [9]. We then scan the VADs with ClamAV Anti-Virus software to identify any potential malware running in the guest environment.

4 Evaluation

We evaluate DACSA on an IBM System X server with a Xeon E5450 Quad-Core CPU with 32GB RAM. Each guest is configured with 1GB RAM and 1 VCPU.

DACSA seeks to enable client VMs as low impact sensors which limit the impact to the guest and host environments. The impact of DACSA on the guest environment is 1) the time the VM is stopped to create the logical copy and 2) the observed guest machine performance reduction during the snapshot and carving process.

We measure the average time the VM is stopped at 0.2112 seconds with a standard deviation of 0.07359 seconds. This time may not be distinguishable from other working latency (e.g. WAN latency, I/O bound pro-

Table 2: Host Percent Utilization. Note: CPU Util. for All VMs is CPU cost per snapshot.

VMs	CPU Utilization			Memory Utilization		
	Idle	1 VM	All Vms Snapshot	Idle	1 VM	All Vms Snapshot
1	0.24	1.89	1.7	10.3	10.31	10.34
2	0.33	2.07	1.53	13.8	13.81	13.86
3	0.95	2.25	1.423	17.37	17.39	17.42
4	1.19	2.36	1.362	20.93	20.95	21
5	0.94	2.66	1.366	24.5	24.55	24.62
6	1	2.2	1.4	28.16	28.18	28.24
7	1.03	2.3	1.385	31.76	31.77	31.67
8	1.29	2.53	1.445	34.85	34.74	34.69
9	1.24	2.51	1.333	37.84	37.64	37.63
10	1.68	2.87	1.266	40.89	40.72	40.69
11	1.04	2.44	1.236	43.87	43.66	43.54
12	1.46	2.55	1.175	46.84	46.72	46.59
13	1.89	2.84	1.168	49.84	49.67	49.5
14	1.83	3.23	1.13	52.75	52.65	52.59
15	1.26	2.6	1.102	55.83	55.65	55.52

Table 1: Guest Performance Impact (Idle VMs used as baseline)

VMs	CPU Ops/Sec		Memory Ops/Sec	
	1 VM	All VMs Snapshot	1 VM	All VMs Snapshot
1	101.737	100.364	98.4414	98.9737
2	98.6734	96.4428	100.049	98.6737
3	98.8032	96.6209	99.3542	97.7142
4	96.4023	96.0401	98.7704	98.9995
5	95.9635	97.6089	97.9539	97.0413
6	98.9038	97.1004	97.5546	97.4161
7	102.878	99.4367	99.1186	98.6395
8	98.8272	96.0313	98.5701	98.8463
9	99.7351	98.6273	98.8986	98.8661
10	97.905	96.0243	98.3499	98.9277
11	97.0239	97.5519	98.9141	97.3235
12	99.5379	97.6411	100.976	100.811
13	98.0907	94.4631	100.327	99.1737
14	100.227	93.9003	98.9583	97.6079
15	96.7603	96.5923	100.306	98.6042

cesses, etc.). However, applications that are latency sensitive should be aware of the potential impacts.

To measure the impact of DACSA on the guest, we run NovaBench in the VM while snapshotting is performed. We incrementally start VMs to observe the impact of DACSA under different host loads. We record the difference between snapshotting and carve a single running VM (i.e. "1 VM") and all running VMs on the host (i.e. "All VMs Snapshot"). The carving process simply identifies the running processes in the guest VM. Table 1 contains the performance impact on the guest CPU and Memory compared to an idle VM baseline. CPU performance is based on the number of integer operations per second while Memory is measured on Read/Write operations per second. We observe the maximum CPU impact is $\approx 6\%$ while maximum Memory impact is $\approx 4\%$. The observed impact is only during the snapshot and carving process. The actual impact to guest VMs is dependent on the length of time required to complete the carving process. This is due to the fact that the OS manages the copies of COW memory pages while the logical copy is held in memory. We note the impact observed while snapshotting one VM versus all VMs is the same allowing scalability across the host.

We measure the impact on the host by observing the CPU and Memory utilization during the snapshot process. Again, we incrementally start VMs and take snapshots and carve process list information of a single VM and all running VMs. Results are found in Table 2. The host memory utilization has no noticeable impact due to Copy-on-Write. The CPU impact during a single snapshot is $\approx 2\%$. The "All VMs Snapshot" depicts the CPU cost per VM snapshot. During the carving process, the executing VM will modify memory which will causes the host to copy pages and increase the write working set. We measured the rate at which the write working set increases at 100-300 MB per minute. Carving typically takes on the order of seconds to extract necessary information. However, we note the time required to carve is dependent on the type of information being extracted for analysis.

We evaluated the correctness of the carving components by extracting all virtual memory for all processes from one logical copy. We then wrote the logical copy to disk using the slow QEMU supported memory snapshotting. The snapshot on disk was loaded with Volatility [2] and we extracted the virtual memory of all processes. We then bitwise compared each memory segment with the memory extracted virtual memory Volatility. We identified all identical copies from both Volatility and our tool.

We then ran the malware, Cerberus Remote Administration Tool, in a VM and infected an Internet Explorer process. After which, we acquired the machine context of the infected machine, extracted the memory segments, and scanned each segment using ClamAV. ClamAV correctly identified the infected Internet Explorer process.

5 Related Work

Performing Out-of-VM analysis is not a new idea [6, 10–12]. For example, VMWatcher proposes running a virus scanner on VM files while the system is running [12]. Unfortunately, VMWatcher tightly couples the guest environment and analysis preventing scalability. Revirt decouples analysis and monitoring by recording the guest VM operations [11]. Analysis is then done by replaying the execution. However, Revirt adds up to 58% overhead to the monitoring VM environment. DACSA adds 4% only during the snapshot and extraction phase which is only in the order of seconds. Similar to Revirt, Aftersight seeks to decouple analysis from monitoring by recording and replaying execution [7]. However, Aftersight suffers from large log files which are not suitable for multiple co-located VMs. Virtuoso allows Out-of-VM utilities to be run on live VM memory with minimal overhead [10]. However, Virtuoso requires extensive training to be done on the target VM to generate Out-of-VM utilities. Furthermore, training on the VM must be redone after each operating system update or internal structure change. Blacksheep clusters Window's Kernel features to identify new rootkits [6]. However, Blacksheep is not suitable for production environments as guest machines are stopped for minutes while execution context is captured. Trend Micro has released a product called Deep Security which provides "agentless" monitoring of VMs for malware and file integrity checking [1]. Unfortunately, the product does not seem to deal with VM memory but only file system interactions. Furthermore, Trend Mirco has not provided an evaluation of the impact of Deep Security on VM performance.

DACSA's approach to fast memory snapshots has similarities to research and techniques into VM migration and cloning. Clark et al. presented Live VM migration where VM's running state was copied to a new machine while the VM continues to run. The process of copying can take upwards of 90 seconds wherein the VM observes performance degradation in throughput up to 20% [8]. DACSA sees less than 5% for only a few seconds. Sun et al. proposed fast VM cloning based on Xen using COW to write protect pages from the hypervisor [14]. The paper discusses the technical challenges of duplicating and sharing VM memory on a clone operation. DACSA sidesteps these challenges by leveraging the process level COW features of forking.

6 Recognition

This work is supported by U.S. National Science Foundation (NSF) under grant 0910767, the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI), and an IBM Open Collaboration Faculty Award.

We would like to thank the reviewers for their insightful comments and critiques. We would especially like to thank Eric Eide for his guidance on revisions and insight into existing literature.

7 Conclusion

In this work, we propose DACSA a decoupled Out-of-VM cloud security analysis architecture. DACSA leverages VMs as low impact sensors enabling virtualized cloud infrastructure to be leveraged as a security analysis testbed. DACSA uses fast snapshots to acquire the running context of VMs with minimal impact to client operation. Existing memory forensic techniques can then be used to extract security centric information from running VMs. Analysis of the extracted information can then be performed in an offline fashion to limit impact to the host and VM environments. Our preliminary results demonstrate DACSA to be a promising approach towards low-intrusive, scalable security analysis for the cloud.

References

[1] Agentless security. Trend Micro.

[2] The volatility framework: volatile memory artifact extraction utility framework. Volatile Systems.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 2010.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[5] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005.

[6] Antonio Bianchi, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Blacksheep: detecting compromised hosts in homogeneous crowds. In *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[7] Jim Chow, Tal Garfinkel, and Peter M Chen. Decoupling dynamic program analysis from execution in virtual environments. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 1–14, 2008.

[8] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[9] Brendan Dolan-Gavitt. The vad tree: A process-eye view of physical memory. *Digit. Investig.*, September 2007.

[10] Brendan Dolan-Gavitt, Tim Leek, Michael Zhivich, Jonathon Giffin, and Wenke Lee. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, 2011.

[11] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th symposium on Operating systems design and implementation*, 2002.

[12] Xuxian Jiang, Xinyuan Wang, and Dongyan 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.

[13] Abhinav Srivastava, Himanshu Raj, Jonathon Giffin, and Paul England. Trusted vm snapshots in untrusted cloud infrastructures. In *Research in Attacks, Intrusions, and Defenses*, pages 1–21. Springer, 2012.

[14] Yifeng Sun, Yingwei Luo, Xiaolin Wang, Zhenlin Wang, Binbin Zhang, Haogang Chen, and Xiaoming Li. Fast live cloning of virtual machine based on xen. In *High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on*, pages 392–399. IEEE, 2009.