Increasing Attacker Workload with Virtual Machines

Stephen Kuhn and Stephen Taylor[1]

Thayer School of Engineering at Dartmouth College

Stephen.kuhn@dartmouth.edu, Stephen.taylor@dartmouth.edu

TR11-002

Abstract:

Much of the traffic in modern computer networks is conducted between clients and servers, rather than client-to-client. As a result, servers represent a high-value target for collection and analysis of network traffic. The observe, orient, decide, and act (OODA) loop for network attack involves *surveillance*, to determine if a vulnerability is present, selection of an appropriate exploit, use of the exploit to gain access, and *persistence* for a time sufficient enough to carry out some effect. The time spent in surveillance and persistence may range from seconds to months depending upon the intent of the attack. This paper describes a novel hypervisor technology that increases attacker workload by denying the ability to carry out surveillance. It also denies persistence, even if the attack is successful and never detected.

Introduction:

The challenges faced by network defenders in protecting systems from malicious code increase daily. The recent SANS vulnerability trends report showed that application vulnerabilities surpassed operating system vulnerabilities, specifically noting that attacks against web services account for over 60% of observed attacks (1). The proliferation of advanced web programming languages, php, javascript, and ruby has increased the complexity of traffic analysis. This increased complexity is compounded by a massive increase in traffic volume from streaming media and peer-to-peer traffic. These factors complicate the task of distinguishing malicious traffic from benign traffic.

Offering assured and available internet services is a key challenge faced by, service providers, the military, and corporations. These services include web hosting, file storage, remote software access, and central database access. As a result of the concentration of information at servers, they are a primary target for adversaries to exploit, forcing providers to expend considerable resources protecting them. Typical defense mechanisms use a combination of intrusion detection systems (2), firewalls (3), and virus scanners (4,5) at the network level. In corporate and military environments additional host base systems are commonly deployed such as virus scanners (4,5), root-kit detectors(6)(7)(8) and more recently website application firewall software (9)(10). These technologies share a common approach using signature based detection methods. Unfortunately, these methods have limited capability to prevent infection from previously undetected malware. As a result, persistent malicious code may never be detected. Although anomaly detection technologies exist to fill the gap, not all anomalous events are malicious, and not all malicious events are anomalous (11).

The challenge of internet addressing and administrating Domain Name Services or DNS, leads most systems administrators to assign static addresses to their servers. Static address assignment allows DNS servers to maintain long update cycles, preventing services from appearing offline and unnecessary traffic. Since a DNS

[1] This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8750-09-1-0213. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

server caches previous address lookups, there is a significant performance advantage from a server remaining in place. Unfortunately this presents a static observable target for adversaries to analyze using network scanners such as NMAP and NESSUS. This surveillance process provides a roadmap to the available vulnerabilities and allows appropriate exploits to be isolated or developed. After access is gained persistence on the static target allows stable reentry point to carry out effects.

The combination of stationary targets, undetectable malicious software, and the small timescales over which a host is exploited allows the attacker to operate inside the defenders OODA loop. This paper presents a novel network hiding technology that moves servers around a network to deny surveillance and periodically reconstitutes them to deny persistence. The goal is to increase attacker workload to the point where the timeliness of attacks is significantly longer than the timescale of day-to-day operations, making attacks irrelevant even if they are successful and never detected.

Related work:

Providing secure and reliable web services is a significant challenge. The most cost effective way to engineer a reliable system from unreliable components is to use *replication*. This approach is used in several current and past systems for web server redundancy and availability. Web service providers can achieve this reliability by leveraging either front end transport or switching at the application layer (12)(13)(14)(15), or replication across the wider internet using a content distribution system such as that provided by Akamai (16).

While the existing replication technologies provide acceptable solutions to the reliability challenge they are not without problems. One example is the use of a round robin DNS approach where several servers host the same content (17). As new connections arrive, they are recycled through the list of available servers. If one of the servers in the queue fails then the clients attempting to connect with it will fail to reach the intended website.

Virtualization research came into existence during the late 1950's (18,19) and has been widely used in the business class server market ever since. Virtualization presents an abstract interface to the underlying hardware for operating systems as illustrated in figure 1. This allows the hardware to be shared among several guest virtual machines.

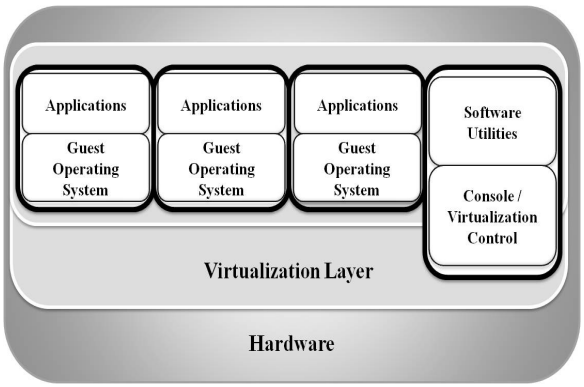

Figure 1. Hypervisor Architecture.

Virtualization has recently gained entrance into the consumer and medium sized business market with VMware's breakthrough research in binary translation (20). Intel and AMD have released hardware support and acceleration that obviates the need for binary translation. These hardware advancements have allowed the open source community to develop competing technologies such as Xen (21) and KVM (22). The open source nature of these systems has made virtualization a popular platform for security research. As virtual machines are separated from the underlying hardware, researchers are able to analyze malware from the relative safety of the hypervisor (23). Others have sought to use the virtualization to provide a clean slate approach that prevents persistent infection of client machines. This approach generates a new virtual machine each time an application is executed preventing persistence in the application code base (24,25).

Network Hiding:

Our proof-of-concept network hiding provides web services through Apache and is based on server *relocation* and *reconstitution* operations. The server's location is periodically migrated within the local enclave IP space and into alternative enclaves using multiple network cards. This has the effect of presenting a moving target to adversaries, while maintaining connectivity to local clients. It has the effect of increasing attacker workload associated with *surveillance*. Leveraging KVM hypervisor technology, the server is repeatedly reconstituted to a fresh service state, potentially using a *different* operating system, akin to a full system reinstall. This has the effect of changing the attack surface while ensuring that malicious code is removed without attempting to detect its presence. This reconstitution denies persistence over long time-scales while falsifying any existing surveillance information that the attacker may somehow

have garnered. These operations can be carried out at random intervals and times. This presents a completely non-deterministic view of the network structure from outside an organizations local area network, while maintaining availability within it.

The approach presents several technical challenges, in particular how to:

- control server relocation and reconstitution,
- preserve connectivity with existing clients, and
- advertise service so that currently unconnected, but authorized, clients are able to locate and connect to the server.

Recall that each server can be assigned one out of several network interface cards (NIC's), and each NIC is connected to a separate logical enclave assumed to be behind a unique firewall/proxy server. Each logical enclave has a DHCP server that serves IP addresses randomly from a large non-routable IP space, orders of magnitude larger than the number of hosts located in the enclave.

Figure 2 outlines the non-deterministic process used to provide network hiding. The process is implemented through KVM hypervisor commands to *define, start*, *undefine* (or remove), and *destroy* (or terminate execution) virtual machines. A programming interface to these commands is provided through the Linux virtual machine library *Libvirt* (26). The library also provides configuration and control of network interfaces. The key feature of this design is that even though the servers presence moves around the physical network by switching network interfaces, all of the transitions occur on top of a single hypervisor. The process begins when the hardware is restarted. The hypervisor enters the ***Start*** state and a *primary* virtual machine ***P*** is created using a random operating system, chosen from a set of pre-configured base-images. An appropriate web server is then bootstrapped on top of the operating system. Our proof-of-concept uses Ubuntu and Fedora with Apache. A random NIC card (enclave) is chosen to operate on, a random IP address is obtained from the associated DHCP server. The network services for the primary virtual machine are then initialized with these properties.

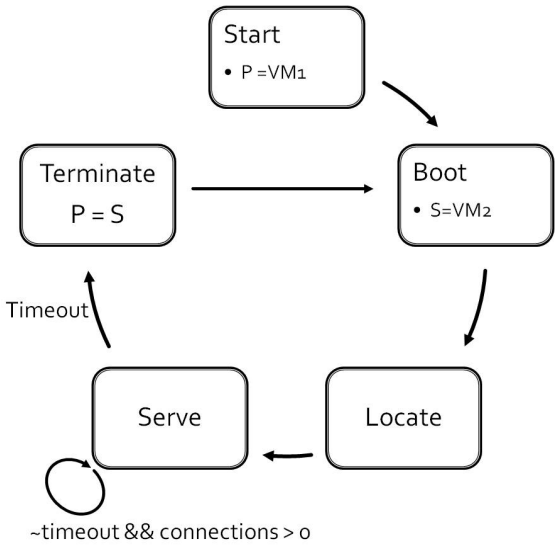

Figure 2: Regeneration Process

The technology subsequently cycles through four primary states:

- ***Boot***: A *secondary* virtual machine ***S*** is created in the background with a different operating system, web server, IP-address, and MAC address (NIC card). On multi-core systems this operation has negligible impact on the performance of the primary virtual machine.
- ***Locate***: A private DNS server is notified of the network address of the primary server. This DNS server can be used only by authenticated clients to determine the location of the server.
- ***Serve***: The primary server responds to incoming connection requests and serves web content. During this activity a random timeout is set and a running count is kept of the number of active connections. The active server continues serve as long as the timeout has not expired and there are active connections.
- ***Terminate***: If the timeout expires, the primary virtual machine terminates but only when all existing connections to it close. The secondary virtual machine becomes the primary virtual machine, all new connections are forwarded to the new primary, and the technology cycles to the BOOT state, where a new secondary client is created.

Figure 3 shows the resulting software stack sharing two NIC cards. There may be two active virtual machines at any one time, each serving a portion of the open connections. The restricted DNS service allows only trusted clients to access the new location of the server when it relocates. This adds an additional layer of indirection that attackers must penetrate, *within a finite time*, in order to locate, identify, access, and effect an active server. In figure 3, the first RedHat server is a previous primary VM whose timeout has already expired but whose connections have not all closed. The first Ubuntu VM is the current primary, its timeout has not yet expired, it serves new connections, and it has been infected as indicated by the dot. The second RedHat VM is the secondary; it will not become active until the next timeout. Notice that each virtual machine uses a unique network connection.

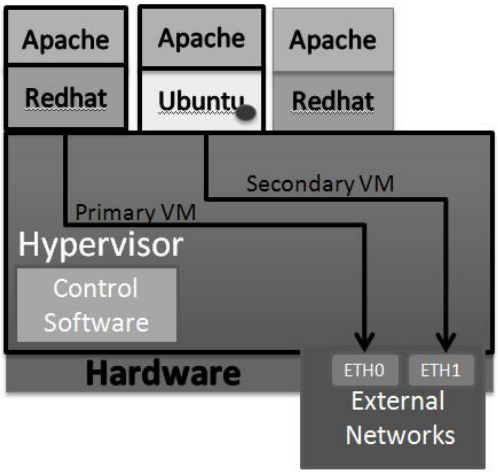

Figure 3: Denying Persistence.

By completely reconstituting the virtual machine hosting the web server, any possibility that malicious code could remain in the system is removed. We prefer this method as opposed to a socket migration technique, because it precludes reintroduction of malicious code from a transferred state.

Unfortunately, dynamically creating network connections inside the hypervisor is not as straightforward as indicated in Figure 3. In order to ensure that any new virtual machine can use any NIC card in the pool and prevent an attacker from observing the server location in one network from another, it is necessary to completely separate the LAN segments. Libvirt provides the ability to generate a virtual bridge inside the hypervisor to effect this separation. Figure 4 refines Figure 3 to illustrate this additional functionality.

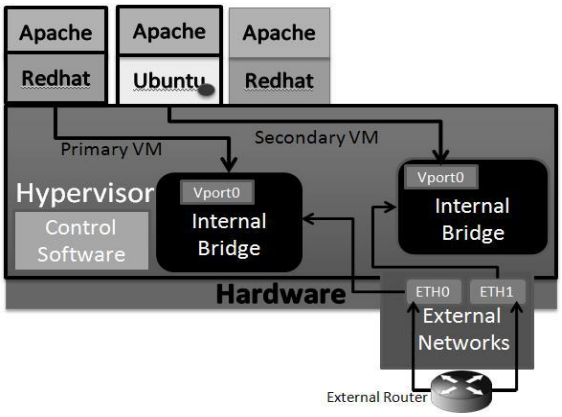

Figure 4: Dynamic Network Isolation.

The virtual bridges mimic the classical local area networks formed by connecting several computers to a physical bridge. One bridge is created and assigned to each physical NIC card. Any virtual machine may be dynamically connected to any physical NIC; this is achieved by assigning the virtual machine a virtual port on the bridge associated with the NIC. The external networks are assumed to be connected via a router hosted outside the hypervisor allowing remote clients to communicate with any active virtual machine.

Observing the enclave structure externally, as an attacker would, the topology appears as multiple distinct Local Area Networks (LANS's) connected via a central router as shown in figure 5. In reality the physical implementation is a single server with multiple connections as shown in figure 4.

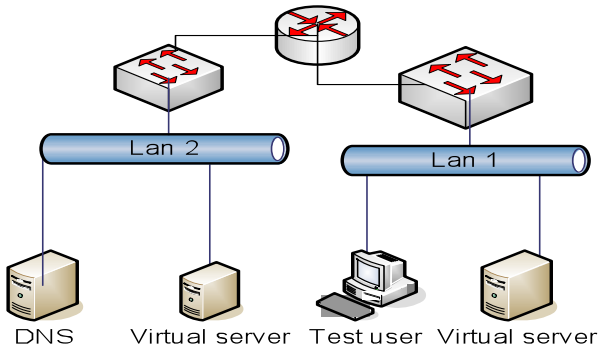

Figure 5: Logical Network Topology

Consider an attacker had previously targeted the present server based upon its IP and MAC address, server type and version. After each reconstitution this information is no longer valid. The attacker must now re-enter his OODA loop and reorient to the new target address

because previous surveillance is now incorrect. Even if the server had previously been infected this implant is no longer present and cannot be contacted or tasked.

Private DNS.

Recall that the regeneration process requires a private DNS server to allow clients to locate a server after it has migrated around the network. Beyond the simple lookup of IP-addresses, the latest DNS implementations, such as BIND (27) provide enhanced functionality for dynamically configuring name references. This involves the update of two components of the DNS record: the CNAME, describing translations between names, and the 'A' records, translating names to IP addresses. Each virtual server must be uniquely named, however, from the viewpoint of external clients, all servers must be accessed through a single consistent name. The CNAME reference system allows this aliasing by creating a name that references another name. The result is a two step process for resolving generic service names, such as www.myserver.org. The DNS server resolves the alias from www to the unique server name and then resolves the unique server name and returns the numeric IP address.

Dynamic DNS or DynDNS provides the control software to dynamically push updates to a DNS server when a new address is selected. DynDNS was originally invented to support home users on modems whose addresses frequently change (28). Previously the only modality for effecting updates required modifying the static configurations of the BIND/DNS server and fully restarting the system, causing a loss of service. Applying updates through the DynDNS mechanism allows us to maintain a consistent server presence, while virtual machines and their associated IP addresses may change.

Lessons Learned:

Connection Migration. Hot swapping network connections between virtual machines presents a significant challenge. Connections to an existing web server cannot be migrated to another web server without causing a connection reset. To understand the problem we categorize the types of network connections, with respect to web servers, into three primary classes:

- Static pages of plain html.
- Streaming pages, containing video or file transfers.
- Stateful pages, such as server side applications that maintain state on the server. For example, shopping carts.

Static web pages are inherently handled by the client's web browser. If the client accesses a page during a VM reconstitution, the client automatically re-requests the page. In our experiments, no perceived impact was observed to the user experience. If the web server is streaming content, as is the case with a file download, the hypervisor cannot simply break the connection; the client would have to resume downloading from the beginning. The same is true with stateful content. The new server would not have the same state and this would cause consistency errors.

One solution is connection state migration in which the active connection is moved between virtual machines in real-time. This presents many challenges, as it requires modification of both the TCP subsystem in the Linux kernel and the web server itself. The MIT CSAIL lab conducted early research in this area (13). The crux of the problem is recreating the state of both the TCP stack, and the web server. To fully migrate the connection and state of the web server would require close coordination between the network layer and application layer, effectively bridging the isolation principles of the layered network design. In addition, copying the state to the new VM would provide the opportunity to carry implants from one VM to another.

Recall that the solution presented here redirects all new incoming requests to the next web server and maintains the previous server until all connections are closed. As a result, streaming connections are not broken and stateful connections are retained for a designed time, limited by the migration timeout.

Dynamic Port Assignment. The initial attempts at denying surveillance exposed a severe limitation of the KVM/Libvirt hypervisor instantiations. The configuration of the hypervisor, the virtual bridges/switches, and the DHCP servers under the hypervisor's control cannot be dynamically configured. Attempts to migrate the server around the network IP space produced erratic behavior, occasionally clients could not connect, other times the migration was seamless. The behavior was caused by the internal configuration of the network infrastructure restarting, occasionally a new client would connect at just the right moment rebuilding the connection, other times not.

The solution to this problem involved dynamic generation of bridges inside the hypervisor to implement the concept illustrated in figure 4. Each time a new virtual machine must be attached to a NIC card,

instead of simply connecting to a bridge port, the old bridge associated with the NIC is removed and replaced with a completely new one. This achieves the concept but is more difficult to implement.

Detecting Connection State. Recall that network hiding requires the ability to detect when all connections to a virtual machine have closed. With few exceptions, such as stateful firewalls, network equipment does not store information about the traffic which passes through it. Similarly the hypervisor does not maintain state for every aspect of the virtual machines it hosts. Connections are passed through the software bridge in a hypervisor just as in the physical world. Consequently, the bridge does not maintain state. One option would be to modify the implementation of the bridge to count open connections. However, a Linux virtual server already maintains connection state, including the number of active connections, in the TCP directory associated the system directory /proc. Unfortunately, this is not generally readable by the hypervisor since the file system of a virtual server is not mounted.

The virtual machine disk is a raw binary file when viewed from the hypervisor. In consequence it can be mounted by the hypervisor as simple another disk. Although it is possible to read the blocks from this disk directly, the encoding format of the disk requires accessing the disk from its root node. Without rebuilding this file system structure there is no simple way to identify the /proc directory we seek to access. To solve this problem there is an open source library *fuse* (29) that allows virtual machine file systems to be mounted and accessed inside a hypervisor.

Unfortunately, the 'files' in the /proc directory are not traditional files; they exist only at the moment they are read in order to ensure that they reflect the most recent system state. When the /proc/tcp6 file containing connection information is accessed, Linux reads the network state and generates the file dynamically, returning the most up-to-date information to whoever accessed the file. The *fuse* library mounting the file system receives the state of the files the moment it accesses them, subsequent reads of the configuration files then have no change. Relying on outdated connection state would lead us to disconnect clients.

In order to solve this problem a module was developed that monitors the state of the connections inside the virtual server and replicates the information in a traditional file accessible from the hypervisor through the fuse library. Once the connection state is accessible it is possible to monitor the number of active connections as needed by the network hiding process.

System Administration. An added benefit of the reconstitution process is that it accommodates testing and deployment of server upgrades independent of the active virtual machine. Upgrades can be tested independently and rolled into the reconstitution queue at any time. This provides a seamless upgrade path for security patches, bug fixes, and configuration changes.

Conclusion and Future Work:

This paper describes a proof-of-concept KVM-based network hiding technology for servers. This architecture has been implemented in the context of web-servers, but equally applies, and can be adapted to, many other services. It increases attacker workload by denying the ability to persist for any meaningful length of time on an infected server. It also denies surveillance by non-deterministically moving a server around the physical network while changing operating system and network properties. The primary challenges involved maintaining control, providing connectivity for active users, and directing new clients to the existing server location. Our solution provides uninterrupted service while effectively increasing attacker workload. Further our solution provides an enhanced capability to upgrade offline machines with patches and upgrades without disrupting the current state.

Implementation of the concepts presented here was significantly complicated by the overall complexity of KVM and Linux. A specialized hypervisor that supports only the operations required to deny surveillance and persistence might provide more direct and natural access to core system parameters making several of the libraries mentioned here irrelevant. Much of the complexity was derived from inconsistent support for hot-swapping of network cards across multiple operating systems. In addition, difficulties arising from lack of dynamic configuration support for networking in the existing LibVirt implementation. As a result, we expect that over time the network hiding process described here will become easier to implement and support in KVM.

References:

1. Dhamankar R, Dausin M, Eisenbarth M, King J, Kandek W, Ullrich J, Skoudis E, and Lee R. Top cyber security risks. from http://www.sans.org/top-cyber-security-risks/ .

2. Scarfone K, Mell P. Guide to intrusion detection and prevention systems (IDPS). 2007; 800-94: .

3. Garfinkel S, Spafford G. Web security \& commerce. Sebastopol, CA, USA: O'Reilly \& Associates, Inc. 1997: .

4. McAfee – antivirus, encryption, DLP, IPS, firewall, email security, web security, SaaS, risk and compliance solutions. 2010, from http://www.mcafee.com/us/ .

5. Norton security - antivirus software | norton store. 2010, from http://buy.norton.com/estore/mf/landingProductFeatures .

6. Quynh NA, Takefuji Y. Towards a tamper-resistant kernel rootkit detector. 2007; 276-283.

7. Anti-rootkit | free rootkit removal | rootkit detection - sophos. 2010, from http://www.sophos.com/products/free-tools/sophos-anti-rootkit.html .

8. RootkitRevealer. 2010, from http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx .

9. Desmet L, Piessens F, Joosen W, and Verbaeten P. Bridging the gap between web application firewalls and web applications. 2006; 67-77.

10. Barracuda web application firewall - web application protection against hackers and vulnerabilities. 2010, from http://www.barracudanetworks.com/ns/products/web-site-firewall-overview.php .

11. Tan KMC, Killourhy KS, and Maxion RA. Undermining an anomaly-based intrusion detection system using common exploits. 2002; 54-73.

12. Deering S, Estrin D, Farinacci D, Jacobson V, Liu C, and Wei L. An architecture for wide-area multicast routing. 1994; 126-135.

13. Snoeren AC, Andersen DG, and Balakrishnan H. Fine-grained failover using connection migration. 2001; 19-19.

14. Cisco systems. failover configuration for LocalDirector. from http://www.cisco.com/en/US/products/hw/contnetw/ps1894/products_configuration_example09186a00800942a8.shtml .

15. Brisco T. RFC 1794: DNS support for load balancing. 1995; .

16. Akamai technologies, inc. from http://www.akamai.com .

17. Round robin DNS load balancing. May 20, 2004. from http://www.content.websitegear.com/article/load_balance_dns.htm .

18. Goldberg RP. Survey of virtual machine research. IEEE computer Magazine 1974; 7: 34-45.

19. Creasy RJ. The origin of the VM/370 time-sharing system. IBM Journal of Research and Development 1981; .

20. Lo J. VMware and CPU virtualization. http://download3.vmware.com/vmworld/2005/pac346.pdf .

21. XEN. from http://www.xen.org .

22. KVM. from http://www.linux-kvm.org .

23. Nguyen AM, Schear N, HeeDong Jung, Godiyal A, King ST, and Nguyen HD. Computer Security Applications Conference, 2009.ACSAC '09.Annual title={MAVMM: Lightweight and Purpose Built VMM for Malware Analysis 2009; 441.

24. Wang J, Jajodia S, Huang Y, and Ghosh A. On-demand virtual work system. pending; .

25. Jiang W, Yih H, and Ghosh A. SafeFox: A safe lightweight virtual browsing environment. System Sciences (HICSS), 2010 43rd Hawaii International Conference on title={SafeFox: A Safe Lightweight Virtual Browsing Environment 2010; 1.

26. Libvirt: The virtualization API. 2010, from http://libvirt.org/ .

27. Liu C, Albitz P. DNS and BIND. O'Reilly Media, Inc. 2006: .

28. P. Vixie E, S. Thomson, Y. Rekhter, and J. Bound. RFC 2136: Dynamic updates in the domain name system (DNS UPDATE). 1997; .

29. FUSE file system in user space. from http://fuse.sourceforge.net .