CyberMoat: Camouflaging Critical Server Infrastructures with Large Scale Decoy Farms

Jianhua Sun
College of William and Mary
jianhua@cs.wm.edu

Kun Sun
George Mason University
ksun3@gmu.edu

Qi Li
Tsinghua University
qi.li@sz.tsinghua.edu.cn

***Abstract*—Traditional deception-based cyber defenses often undertake reactive strategies that utilize decoy systems or services for attack detection and information gathering. Unfortunately, the effectiveness of these defense mechanisms has been largely constrained by the low decoy fidelity, the poor scalability of decoy platform, and the static decoy configurations, which allow the attackers to identify and bypass the deployed decoys. In this paper, we develop a decoy-enhanced defense framework that can proactively protect critical servers against targeted remote attacks through deception. To achieve both high fidelity and good scalability, our system follows a hybrid architecture that separates lightweight yet versatile front-end proxies from back-end high-fidelity decoy servers. Moreover, our system can further invalidate the attackers' reconnaissance through dynamic proxy address shuffling. To guarantee service availability, we develop a transparent connection translation strategy to maintain existing connections during shuffling. Our evaluation on a prototype implementation demonstrates the effectiveness of our approach in defeating attacker reconnaissance and shows that it only introduces small performance overhead.**

I. Introduction

Recent years have witnessed the explosion of targeted attacks against the critical server infrastructures within both government organizations and businesses, and the number of data breaches increases tremendously with the continuous proliferation of exploitable zero-day vulnerabilities [1]. Particularly, an advanced persistent threat (APT) may enable an unauthorized attacker to bypass the traditional security measures such as firewalls and intrusion detection systems (IDS) and stealthily gain access to the sensitive data on the victim servers.

Consequently, deception-based techniques have re-emerged as an additional line of defense to supplement the traditional preventive security measures [2], [3]. Instead of relying on monitoring abnormal attack patterns, deception-based techniques use advanced luring techniques and engagement servers to entice an attacker away from the valuable servers. As the average time to identify and resolve a data breach for malicious attacks is 82 days [4], specially crafted *decoys* can be utilized to help shorten the detection and prevention cycle. Specifically, a decoy provides a carefully isolated environment for misdirecting attackers and feeding them disinformation in the form of falsified data such as encryption keys, database entries, and OS fingerprinting information [5], [6].

Most existing deception-based defenses adopt traditional *honeypot* technology [7], [8] to develop decoy systems for attack detection and information gathering [9], [10], [11]. However, their effectiveness has not met the expectations of the security practitioners due to two major limitations. First, the number of believable decoys is constrained by the limited system resources. There is a trade-off between the decoy fidelity and the required system resources. Low-fidelity decoys require less system resources but may be easily identified by attackers. In contrast, high-fidelity decoys may not be easily identified by attackers but require more system resources to better mimic the real server. For instance, previous research efforts have designed virtual machine (VM) based honeyfarms [12], [13]. However, their design either assumes that there exist only short term connections towards a limited number of IP addresses and thus unable to handle large numbers of simultaneous attacker requests, or requires the availability of abundant resources for hosting large numbers of virtual machines. Second, decoys are statically deployed. With sufficient support, APT attackers may finally identify all static decoys through either timing-based or fingerprinting-based analysis [14], [15]. It is critical to design believable, scalable, and dynamic decoys to tackle these two challenges.

In this paper, we design a decoy-based deception mechanism named CyberMoat that can protect critical servers against targeted attacks with a large number of high-fidelity decoys, which can be dynamically created and managed using limited system resources. To relieve the tension between decoy fidelity and scalability, we adopt a hybrid architecture that separates an extensive layer of front-end proxies that focus on network stack processing from back-end decoy servers that serve the service requests. Since the size of the proxies is extremely small, we can create a proxy pool that consists of hundreds (or even thousands) of lightweight proxies, which transparently redirect the network traffic between the attackers (or legitimate users) and the back-end decoy servers (or protected servers). The decoy servers closely resemble real servers with full-fledged operating system and services but only provide falsified information.

To further defeat APT attackers, we dynamically mutate the proxies' addresses to not only invalidate the attacker's knowledge gained from prior network scanning, but also diffuse the traffic targeting at an overloaded proxy. However, since existing network connections will be disrupted when shuffling the proxy's IP and MAC addresses, legitimate users may suffer degraded user experience. To guarantee service availability, we develop a transparent connection migration strategy so that the previously alive network sessions are not interrupted during proxy address shuffling process.

We implement a CyberMoat prototype that uses

978-1-5386-0683-4/17/$31.00 ©2017

ClickOS [16] as the front-end proxies and uses full-fledged Xen virtual machines as the back-end decoy servers. The proxies redirect connection requests to the back-end decoy servers that install the same set of software stack as the real server. We design a centralized control plane to manage the proxy address shuffling and ensure seamless connection migration. We evaluate the effectiveness of our prototype in disrupting network reconnaissance and show that our system introduces small performance overhead.

In summary, we have made the following contributions.

- We develop a dynamic deception mechanism that protects critical real servers with a large scale high-fidelity decoy farm, which can be dynamically configured to trap and misinform targeted attacks.
- We develop a highly scalable decoy system that combines an extensive layer of proxies with high-fidelity decoy servers. Through this hybrid design we can successfully misdirect the attackers and entice them away from the protected servers.
- We introduce a proactive approach of randomizing the proxy network addresses to invalidate the attacker's reconnaissance efforts. Moreover, we develop a seamless connection migration mechanism to maintain those existing network connections during the randomization process.
- We implement a system prototype of our defensive framework and evaluate its security effectiveness.

II. Background

Unikernels are specially built library operating system (LibOS) style virtual machines providing a dedicated runtime environment for running single purpose applications [17]. By trimming away cumbersome system services and drivers from traditional OS, they can shrink the attack surface and resource footprint of cloud hosted services. Compared to full-fledged virtual machines, unikernels are also featured with faster boot times and optimized performance.

As one unikernel, ClickOS has been employed to build up the network function virtualization (NFV) platform by consolidating heterogeneous middlebox processing onto the same hardware [16]. It runs the Click software router above the Xen [18] based MiniOS and employs system optimizations such as batching and removing unnecessary buffer management to improve the performance. Click provides a modular framework for constructing middleboxes such as firewalls and NAT boxes by reusing the functionality of over 300+ stock elements with a specialized configuration language. The memory footprint of ClickOS images is as small as several MBs and ClickOS virtual machines can boot in tens of milliseconds.

III. Threat Model and Assumptions

We focus on targeted remote attacks that aim at compromising critical real servers such as web servers and database servers, and do not consider "remote insiders" that penetrate the protected network by compromising legitimate users through social engineering. We assume targeted attacks are typically preceded by a reconnaissance phase during which attackers gather valuable information about network topology, operating system (OS), available services, and unpatched or undisclosed vulnerabilities. The attackers can undertake various scanning strategies such as OS and service fingerprinting to surveil the target network and system. OS fingerprinting aims at determining the operating system of a remote host through passive sniffing and traffic analysis, or active probing. Similarly, service fingerprinting aims at determining the categories and versions of the running services.

To keep the attack stealthy and protect the attack strategy, the attackers need to identify the decoy servers from the real servers and avoid attacking those decoys. They can use either fingerprint-based or timing-based techniques to identify a decoy server. First, when the decoy servers do not have the same set of software as the real server installed, the attacker can detect the difference of the running software and system status. Second, since the decoy servers may have constrained system resources, their service response can be slower than the real server.

IV. System Architecture

We design a deception framework called CyberMoat to proactively protect critical online servers against targeted remote attacks. As shown in Figure 1, CyberMoat follows a hybrid architecture that combines an extensive layer of front-end proxies with the back-end decoy servers. It consists of four major components: *CyberMoat control plane*, *proxy pool*, *decoy server farm*, and *CyberMoat gateway*. The CyberMoat control plane monitors and regulates the traffic towards the decoy proxies, manages and shuffles the proxies, and manages the back-end decoy servers. The proxy pool is comprised of hundreds or even thousands of lightweight proxies that transparently redirect the network traffic between the attackers (or legitimate users) and the back-end decoy servers (or protected servers). The decoy server farm contains a small number of high-fidelity decoy servers that process malicious service requests diverted from the front-end decoy proxies. The CyberMoat gateway enforces the traffic forwarding rules configured by the control plane and ensures seamless migration of alive network connections during the proxy shuffling.

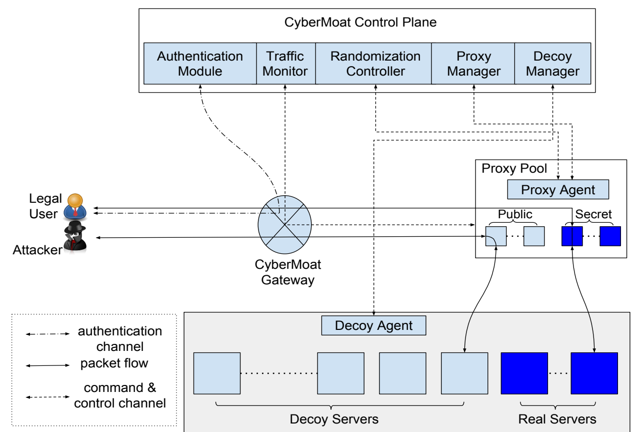


Fig. 1: CyberMoat Architecture

A. CyberMoat Control Plane

The management of the entire framework is integrated into a centralized control plane, which grants us the ease of monitoring the decoy platform and the agility in response to extreme attack scenarios such as large scale network scanning and flooding against the decoys. The control plane performs five core functions, which are elaborated as follows.

Authentication Module. It verifies the identity of the client initiating service requests to the real servers and maintains a white list of authenticated users. Only legitimate users can authenticate themselves, be added into the white list, and then get redirected to the proxy associated with the real server. Given the white list, the CyberMoat gateway enforces that the requests from legitimate users are transparently redirected through some secret proxies to the real servers, while malicious requests are redirected to the decoy servers through some public proxies. Our approach is orthogonal to the usual firewall or IDS based defense, in that we assume sophisticated attackers can evade those defenses and bypass the authentication mechanism. Different from the traditional access control systems, we deliberately allow the suspicious requests from those potential attackers capable of bypassing the authentication to pass through the proxies and then transparently divert the requests to the back end decoy servers. Thus, when suspicious attackers proceed to interact with the decoy servers, they are being trapped to compromise the decoy servers just as they endeavor to breach the real servers.

Traffic Monitor. This module constantly monitors the network traffic towards the decoy proxies to detect the occurrence of potential network congestion. Since the decoy proxies are used to process malicious requests, the traffic monitor will monitor the healthy status of the decoy proxies by tracking the workload and throughput of each proxy and raise alarms when detecting attacks. It provides reactive response by instructing the randomization controller. For instance, when the traffic monitor discovers that a large scale scanning is in progress, it can raise an alarm to the randomization controller so that the proxy address configurations will be shuffled accordingly to invalidate their scanning results.

Proxy Manager. It is responsible for instructing the proxy agent in the proxy pool to spawn a large number of concurrently running proxies, the number of which can be adaptively changed according to the monitored attacker's behavior and the available system resources. It controls the overall topology of the entire proxy network, the specific configuration of each proxy (e.g., IP and MAC addresses), and the functions carried out by each proxy. For instance, the proxy can be programmed to serve as a network address translation (NAT) box that transparently relays traffic, a firewall that actively blocks suspicious traffic, or any popular middlebox with multiple network functions. It can also be configured to directly respond to scanning probes instead of handing over the task to the back-end decoy servers.

Randomization Controller. This module coordinates the dynamic mutation of the network address configurations among the proxies. It listens to the alarm signal from the traffic monitor and determines the algorithm and frequency to randomize the network addresses. It then generates new proxy address configurations according to the chosen algorithm and guarantees that there is no interference in the address assignment. Afterwards, it distributes the new configurations to the proxy agent in the proxy pool and instructs the agent to carry out proxy address shuffling. During the shuffling process, existing network connections can be interrupted due to the changes of the IP addresses. To guarantee service availability, the randomization controller instructs the CyberMoat gateway to translate each alive connection's IP and MAC addresses associated with a proxy into the proxy's new addresses. The translation process is completely transparent to both legitimate users and the attackers, and it does not require any modification to the end hosts.

Decoy Manager. It is responsible for deploying and managing the back-end decoy servers by communicating with the decoy agent in the decoy server farm. Since the number of decoys that can run concurrently is largely constrained by the available system resources, the decoy manager monitors the decoy servers and instructs the decoy agent to recycle the compromised decoy servers and create new ones accordingly.

B. Proxy Pool

The proxy pool includes a proxy agent that receives instructions from the proxy manager and generates hundreds (or even thousands) of lightweight proxies, which transparently relay the traffic between the attackers (or legitimate users) and the back-end decoy servers (or real servers). We classify the proxies into two categories: *secret proxy* and *public proxy*, the partition of which is controlled by the Proxy Manager. Secret proxies are only accessible to the legitimate users who have passed the authentication process and redirect their service requests to the real servers; while public proxies are open to any access request and redirect those requests to the decoy servers.

All proxies present an attacker with the illusion of a large number of co-locating servers, since each proxy is visible to the external users like a real server with its network address publicly advertised. When an attacker attempts to connect to a secret proxy, it will be transparently diverted to one of the public proxies. The proxies may directly generate timely responses to those scanning probes such as ICMP pings or TCP SYN scans to reduce the workload of the decoy servers. In addition, we proactively shuffle the network addresses of the proxies when our system is under persistent and large scale network reconnaissance or a single proxy is targeted for network flooding. In this way, we can invalidate the attackers' knowledge gained from the previous scanning efforts and raise the bar for the attackers to locate the protected server.

C. Decoy Server Farm

A decoy agent receives instructions from the decoy manager to generate high-fidelity decoy servers. The back-end decoy servers cannot be directly accessed from the external network and all accesses have to be redirected by the proxies. Based on the attacker's knowledge of the real server, there are different options for deploying decoy servers. If the attacker knows nothing about the targeted system such as its OS and application versions, we may deploy decoy servers running different software stacks. Thus, the attackers need to collect different sets of exploits to compromise multiple servers before they can further investigate which ones are decoys. We can even deliberately expose some vulnerabilities on certain

decoy servers to attract and trap the attackers. However, when the attackers know the software stacks running on the real server, they can easily filter out those decoy servers that run different OSes and applications. On the other hand, we can run the same set of software stacks on both the real servers and the decoy servers. Thus, the attacker can hardly distinguish a decoy server from the real server by using either fingerprint-based or timing-based techniques.

D. CyberMoat Gateway

CyberMoat gateway receives instructions from the control plane on steering packet flows through the proxies. It carries out three functions programmable by the control plane. First, it relays the normal traffic between the external users and the proxies and responds to the queries from the traffic monitor with the flow and port statistics as well as the status of the proxies. It can even duplicate packets to the traffic monitor through mirroring to detect attacker scanning or other malicious activity through deep packet inspection. Second, it proactively re-configures the forwarding rules for the proxies with changed network addresses due to the shuffling, and avoids any further control plane involvement in processing future packets towards those proxies. Lastly, it assists the randomization controller to migrate the existing network connections to the new proxy addresses by enforcing the dynamically updated address translation rules.

V. System Implementation

We implement a CyberMoat prototype as shown in Figure 2. Both decoy servers and real servers are implemented as Xen-based virtual machines (VMs). The small-sized ClickOS [16] is used to implement the decoy proxies.

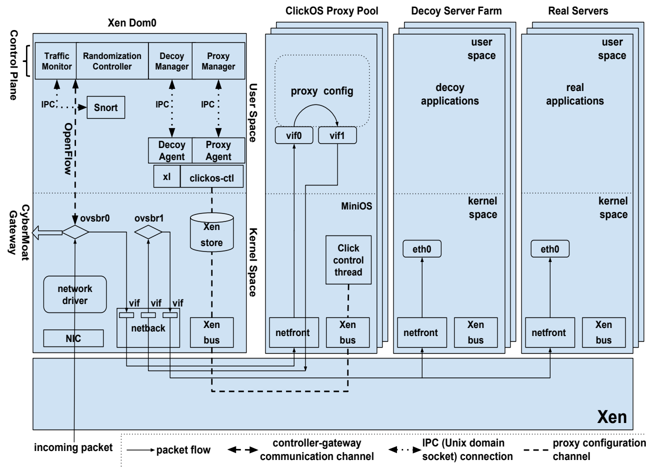


Fig. 2: CyberMoat Prototype Implementation

A. SDN-based Control Plane

We implement the CyberMoat control plane inside Xen Dom0 using the Ryu [19] SDN framework. The authentication module, the traffic monitor, and the randomization controller are all implemented as RYU application modules. In particular, the traffic monitor is integrated with Snort [20]. The authentication module enforces password based authentication and configures `ovsbr0` to redirect the initial service requests to a standalone authentication server process, which stores all user names and passwords in an SQLite database. The proxy manager and decoy manager are implemented as daemon processes using python scripts. Similarly, the proxy agent and decoy agent are also implemented as daemons, and they communicate with the proxy manager and decoy manager through Unix domain socket (i.e., IPC socket) connections.

Proxy Status Monitoring The traffic Monitor sends `OFPFlowStatsRequest` and `OFPPortStatsRequest` queries to the `ovsbr0` gateway periodically to gather flow and port statistics associated with each proxy. It collects the byte and packet count of the traffic traversing each active flow, and the number of packets/bytes transmitted, received, or dropped by each port. By keeping track of the distribution of traffic among the proxies, it can detect any spikes of the proxy workload and identify if a proxy is overloaded. Further, it configures `ovsbr0` to mirror packets to Snort, and watches on a dedicated Unix domain socket for any alert message sent from Snort, which indicates if the proxy network is undergoing large scale scanning such as ping sweep or TCP SYN scan, or a certain proxy is flooded with voluminous packets.

Listing 1 shows an example Snort rule that can detect TCP SYN flood scan, which is defined as any TCP connection requests sent towards our proxy network more than 100 times in 5 seconds, where the actual parameters used can be adjusted. Once either condition is identified, an alarm signal will be raised to the randomization controller, which then initiates the shuffling of the proxies' address configurations to either invalidate the attacker's scanning efforts or reroute the traffic through other proxies to evacuate the attacked proxy.

Listing 1: Snort rule for TCP SYN flood detection

```
alert tcp any any -> $PROXY_NET any (msg:``TCP SYN flood'';
flow:stateless; flags:S; detection_filter:track by_dst,
count 100, seconds 5; sid:1000001; rev:1)
```

Proxy Shuffling We implement the randomization controller to be able to perform both *proactive* and *reactive* shuffling. For proactive shuffling we periodically shuffle the IP and MAC addresses of the proxies with fixed frequency; while for reactive shuffling, we shuffle when the traffic monitor detects large scale adversarial scanning or flooding behaviors. In detail, the randomization controller first computes a randomized assignment of proxy IP and MAC addresses to the switch ports and then issues `OFPFlowMod` instructions to `ovsbr0` to modify packet forwarding rules related to proxies with changed addresses. For example, assuming proxy a with IP address ip is attached to port p of `ovsbr0`, and flow rule f forwards all packets destined for ip to switch port p, if a's address is changed to ip', then f will be modified to forward packets destined for ip' to port p. The newly computed proxy IP and MAC address assignment is exported as JSON messages through a RESTful API to the proxy manager, which then generates new proxy configurations and sends them to the proxy agent to refresh the proxy configurations accordingly.

Seamless Network Connection Migration To maintain

existing network connections during the shuffling process, we develop a seamless connection migration scheme. Consider one alive connection between an external user and a proxy identified by a 4-tuple (src_mac, src_ip, dst_mac, dst_ip), where the source refers to an external user and the destination refers to a proxy. Once the proxy's MAC address is changed from dst_mac to new_mac and its IP address is changed from dst_ip to new_ip after a round of shuffling, the randomization controller inserts into the ovsbr0 flow table a DNAT rule to translate the destination addresses of incoming packets from external world to new_mac and new_ip, and insert an SNAT rule to translate the source addresses of outgoing packets from the proxies back to dst_mac and dst_ip. To avoid potential flow table explosion, we track the establishment and termination of a TCP connection through TCP SYN and FIN flag matching, and only keep NAT rules for those existing connections. Since Ryu does not support direct TCP flags matching, we leverage Nicira [21] extended matching supported by Open vSwitch to capture the initial TCP SYN packet and the last TCP FIN packet for a TCP session. Once a packet with TCP FIN set is captured, the randomization controller removes the corresponding flow rules from the flow table of ovsbr0.

Decoy Replication We integrate both front-end proxy and back-end decoy server replication into the decoy deployment. The proxy manager first generates the proxy configurations, i.e., the IP and MAC addresses as well as the virtualized network functions programmed as Click routers. These configurations are sent to the proxy agent through a dedicated Unix domain socket connection. After that, an entire pool of ClickOS proxies with identical network functions are created. When one round of proxy shuffling is triggered, the proxy manager distributes the newly created configurations to the proxy agent and instructs it to refresh the address assignment. Likewise, the decoy manager controls the deployment of back-end decoy servers via the aid of the decoy agent. The proxy manager determines the category of services (e.g., web servers or database servers) hosted on the decoy servers and issues commands to the decoy agent through a Unix domain socket connection to create decoy server virtual machines (VMs) and install pre-configured services.

B. Virtualized Gateway

We employ an Open vSwitch [22] bridge ovsbr0 to serve as the CyberMoat gateway between the external Internet and the front-end proxy pool. Specifically, ovsbr0 communicates with the control plane following the standard protocol OpenFlow 1.3 [23]. Initially, the control plane broadcasts ARP probes towards the decoy proxy pool and then build up a packet forwarding flow table into ovsbr0. After that ovsbr0 directly forwards incoming packets to the corresponding proxies without any further control plane involvement. Meanwhile, it is configured to mirror packets to the traffic monitor for analysis and generates replies to the flow statistics queries from the traffic monitor. Whenever proxy address shuffling is triggered, its flow table will be re-configured to ensure correct packet forwarding and seamless connection migration. Moreover, we also create another virtual gateway ovsbr1 to connect the proxy pool with the back-end decoy servers or the real servers. It is configured to work in the Open vSwitch normal mode and functions as a regular router to forwards packets.

C. ClickOS-based Proxies

We use ClickOS-based virtual machines to build up the front-end proxies. The ClickOS VM is created using standard Xen *xl* toolstack. The functionality of a proxy is programmed using the Click modular router [24] configurations, and each configuration is installed or unloaded using the ClickOS Control clickos-ctl [25]. Specifically, a special Click control thread is created after the ClickOS VM boots, which will then create an *install* entry in the Xen store. After that, clickos-ctl can be used to install Click configurations into the ClickOS VM by writing to the specific Xen store entry. Finally, the Click control thread will spawn a separate thread to run the installed Click configuration. Indeed, the modular feature of Click makes it flexible in deploying customized network functions. For each proxy, we create an upstream virtual interface *vif0* to connect to the external world and a downstream virtual interface *vif1* to connect to the back-end decoy server. The IP address of the external facing *vif0* is published to both legitimate users and potential attackers, while the IP address of the internal facing *vif1* is concealed and located within a different subnet from the external IP address.

In our prototype, each proxy is programmed with two main functions. First, it can directly generate replies to simple large scale network scanning. For instance, it can directly respond to ping sweep using the ICMPPingResponder Click element. In this way, we can not only isolate the voluminous scanning traffic from the back-end decoy servers so that they are not overwhelmed, but also generate timely responses to mislead the attackers with the illusion of a large number of coexisting servers. Second, it works as a transparent network layer proxy that redirects the packets back and forth between external users and the decoy servers. The proxy's core function is implemented by extending the IPAddrPairRewriter Click element to translate the destination address of incoming packets towards a proxy into the address of the decoy server connected with the proxy's internal interface and translate the source address of outgoing packets from the decoy server into the address bound to the proxy's external interface. The lightweight proxy only runs the necessary elements for running the configured network layer functions, and it does not perform any application layer processing.

The proxy agent performs two main functions. First, upon receiving a command to shuffle the proxies, it uses clickos-ctl to refresh the IP and MAC addresses of the proxies. Particularly, it uses clickos-ctl to remove the old Click router instance from the proxy and then installs a new instance with updated address configurations. To minimize the update time, we allocate the task among multiple threads and use the Xenstore dameon oxenstored that is optimized for concurrent transaction processing. Second, when the traffic monitor in the control plane detects any crashed proxy and notifies the proxy agent, it recycles the corresponding proxy and recreates a new one. Since our proxy is lightweight with a minimalistic operating system, it is fast to instantiate a proxy, which makes our proxy design resilient to proxy failures.

D. VM-based Decoy Server

We implement the back-end decoy servers as full-fledged Xen virtual machines. The decoy manager in the control plane controls the system and service configurations of the decoy servers by instructing the decoy agent to create a pool of decoy VMs from a template decoy VM. In our prototype, each decoy VM possesses system and service configurations that closely resemble real servers, including operating system distro, kernel version, service category and version, etc. A major difference from the real server is that the decoy VM may host services with unpatched vulnerabilities to trap the attackers and feed them with false data. We deploy an Apache HTTP server 2.4.7 and an OpenSSH server 6.6.1p1 in both the decoy VMs and the real servers.

The decoy agent continuously tracks the running status of each decoy VM and reports it to the decoy manager. It monitors if a decoy VM crashes or a session is timed out, which usually indicates successful exploits. When it happens, the decoy agent destroys the victim decoy VM to release the system resource and uses `virt-clone` utility to create a new VM from the decoy VM template.

VI. Experimental Results

Through performing experiments on our CyberMoat prototype, we evaluate our deception system's performance characteristics and its effectiveness in mitigating targeted attacks. All experiments are performed on a laptop with an eight-core Intel Core i7-4712HQ CPU and 16 GB RAM running 64-bit Ubuntu 14.04 (Trusty Tahr) and Xen hypervisor version 4.4.2. Xen Dom0 is configured with fixed 4 GB RAM and pinned with four vCPUs. Each proxy is configured with 8 MB memory. We create a decoy VM, a real server VM, and a test VM, all running 64-bit Ubuntu 14.04 with 1 GB memory. We install OpenSSH server 6.6.1p1 and Apache HTTP server 2.4.7 in both the real server VM and the decoy VM. The test VM is used to emulate real users and initiate connections to the proxies.

A. Performance Evaluation

We first measure the scalability of our system on decoy deployment. Then, we evaluate the performance overhead introduced by the transparent proxy redirection and the proxy address shuffling to estimate the impact of our deception mechanism on the legitimate users.

Decoy platform scalability. To test the scalability of our system, we create proxies sequentially and measure the time taken to boot one proxy and install the click configuration. We consider the booting process of a proxy complete as long as it can respond to ping probe. The results are shown in Figure 3. When creating 300 working proxy VMs with each occupying 8 MB memory, we observe a slowdown. For the first proxy, it takes 0.27 seconds and 0.02 seconds to create the proxy VM and install Click configuration, respectively. However, for the last proxy, it takes 1.34 seconds and 0.36 seconds to finish the booting process. The slowdown is mainly because the Xen toolstack is not optimized for continuously booting a large number of virtual machines. Nonetheless, the experiment results demonstrate the capability of our system for massively consolidating a large number of decoys. Moreover, the small time latency for new proxy creation renders our system agile in the case of proxy crashes.

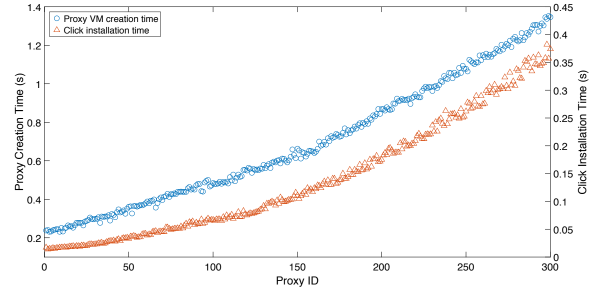


Fig. 3: Time Latency for sequentially creating proxy VMs and installing click configurations.

Proxy shuffling time breakdown. To quantify the responsiveness of our system in reaction to network reconnaissance, we measure the time of proxy shuffling, which involves two operations: modifying the flow rules and refreshing the proxies' address configurations. We write a python script in the test VM to create multiple concurrent TCP connections to the proxies that redirect the connections to the back-end decoy VM. We measure the overall time for re-configuring flow rules and updating proxy addresses when we perform multiple rounds of proxy address shuffling with fixed frequency (i.e., 5 seconds per shuffle in our test). Figure 4 shows the measurement results for up to 200 connections with each measurement repeated ten times. For the case of 200 connections created towards 200 proxies, it takes 0.15 seconds for the randomization controller to modify the packet forwarding and NAT rules configured in `ovsbr0` and less than 5 seconds for the proxy manager to update all the proxies' addresses. We see that the overhead is mainly caused by the proxy address update process, since `clickos-ctl` needs to repeatedly write to the XenStore to remove and install click router instances into the proxy VM. Though we may optimize XenStore implementation to further reduce the time delay, the proxy shuffling time delay in our current implementation is still acceptable as shuffling 200 proxies only takes several seconds.

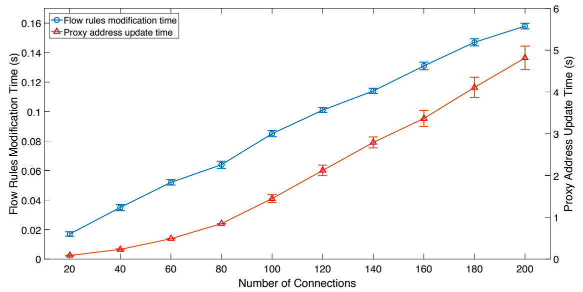


Fig. 4: Time latency for updating switch flow rules and proxy addresses.

Flow redirection and address shuffling overhead. Interposing a transparent proxy layer in the users' packet forwarding path is the main source of performance overhead in CyberMoat. To estimate the impact of flow redirection and proxy shuffling on the users' performance, we first use the Secure Copy (SCP) utility to remotely download files

of varied sizes from the server VM to the test VM. We compare the file transfer performance in three scenarios: (1) The test VM and the server VM are directly connected through an Open vSwitch bridge; (2) They are connected through extra proxy redirection but without proxy shuffling; (3) They are connected through the proxy and periodic proxy shuffling is enabled. The three cases are denoted as *np*, *p* and *ps*, respectively. The results in terms of file transfer throughput are shown in Figure 5a. On average the proxy redirection causes 5% throughput degradation and the proxy shuffling incurs extra 3% overhead. We also investigate the file transfer performance while performing multiple rounds of shuffling with different frequencies and the results are shown in Figure 5b. We can see that the proxy shuffling can be performed as fast as once per 5 seconds while still maintaining the file transfer session, and the overhead incurred by shuffling is only around 3% compared to the case without any shuffling. Moreover, the overhead remains consistent whatever frequency the shuffling is performed.

Furthermore, we use the *netperf* benchmarking tool to investigate the performance overhead in CyberMoat in terms of network throughput, round trip time (RTT), and CPU utilization. We ran the netperf client on the test VM and the netperf server on the server VM. The netperf TCP_STREAM mode is used to measure the aggregate TCP transfer throughput; while the TCP_RR mode is used to measure the round trip time. All the experiments were performed with respect to message sizes in the previously mentioned three scenarios and each measurement continues one minute. Figure 6 shows the measurement results. We can see that without proxy redirection the throughput (around 92 Mbps) approximates line rate for all the message sizes. However, the proxy layer introduces around 4.6% throughput overhead with the sender CPU utilization dropped 30%; while the proxy address shuffling introduces extra 3.2% overhead with the sender CPU utilization dropped another 23%. Similarly, the proxy layer prolongs the average round trip time by 6% and the extra shuffling causes another 3% delay. Both the sender and the receiver CPU utilization drops as the message size increases.

B. Security Evaluation

We also evaluated our system from the perspective of an attacker trying to discover live hosts in the target network and determine the operating system of each remote host or the type of services running on it. We deployed 200 proxies which initially transparently redirect traffic to a back-end decoy server VM configured with an OpenSSH server and an Apache HTTP server. Then we employed the nmap [26] fingerprinting tool to surveil the target network. The scanning process consists of three phases, including host detection through ICMP probe, port discovery using TCP SYN and stealth scan (Xmas scan in this test), and vulnerability assessment through OS and service version detection. By default, it scans the most common 1000 ports for each protocol. All the tests have -T4 option turned on to enable faster scanning. For the vulnerability assessment the -A option is used to enable a composite scan including OS detection, version detection, script scanning, and traceroute. Table I summarizes the scanning results and the average time taken to finish scanning a host. Overall 200 alive hosts are discovered through simple ping probe. Both TCP SYN and Xmas scans can discover 200 alive hosts, each of which is configured with *ssh* and *http* services. The composite scan further identifies the remote OS kernel version (Linux 3.x) and service versions (i.e., ssh protocol 2.0 and Apache httpd 2.4.7 Ubuntu). Moreover, its traceroute scanning only discovers one hop which indicates that our proxy layer is transparent to the attackers. Therefore, our system can deceive an attacker into a belief that a large number of valuable hosts are running, and lures him to invest more resources in compromising the discovered hosts.

TABLE I: Nmap Scanning Results

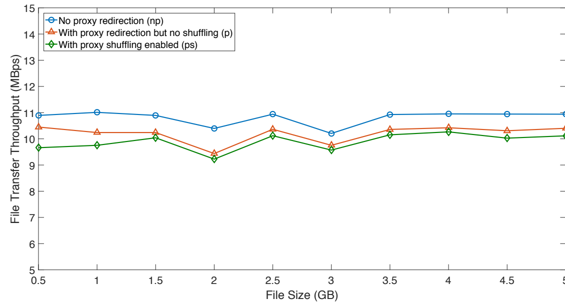
Scan Type	Number of Alive Hosts	Average Time (s)	Services	Port Status	OS Version
Ping	200	0.01	-	-	-
TCP SYN	200	10.18	ssh/http	open	-
TCP Xmas	200	10.76	ssh/http	open|filtered	-
Composite	200	24.35	ssh/http	open	Linux 3.x

VII. Security Analysis

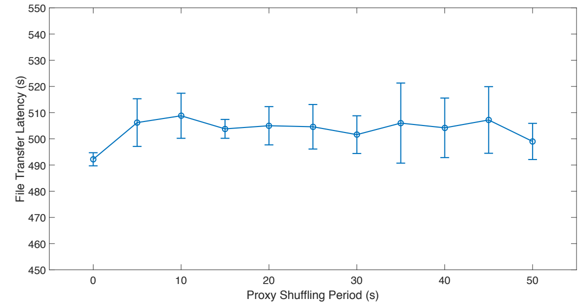
We assume that the attackers need to perform thorough reconnaissance before launching targeted attacks. Using a hybrid architecture consisting of lightweight front-end proxies and back-end decoy servers, our system can present a large number of concurrently running yet high-fidelity decoys to the attackers and mislead them into believing that they have discovered real valuable servers. Moreover, through proactive shuffling our system can further disrupt the attackers' scanning process and force them to invest more resources into the reconnaissance. As demonstrated in our experiment, an attacker needs to spend tens of seconds through nmap scanning to discover the OS and service versions of a target host, whereas we could shuffle the proxies' address configurations in just several seconds to distort the attacker's perception of the target network.

Our system can achieve transparent attacker misdirection as the proxy layer incurs minimal overhead and the decoy server closely resembles real server. Instead of completely filtering out malicious traffic, our defensive mechanism endeavors to distract the attackers' attention away from the real servers, which is orthogonal to regular firewalls and intrusion prevention systems. First, the proxy layer is versatile in terms of functions (e.g. generating responses directly to ping or TCP SYN probes) yet almost transparent. Second, the hosted decoy services can further interact with the attackers and respond to the service requests with falsified but seemingly real information. Moreover, the overhead caused by the thin proxy layer is uniform across the legitimate users and the attackers. Therefore, it is difficult for a malicious user to identify the real server via timing analysis.

Due to the minimal functions in each proxy, its trusted computing base is small and promising to be formally verified. Since each proxy is lightweight with limited memory and processing capability, it creates an opportunity for the attackers to flood a proxy with voluminous traffic to process. Our system can effectively mitigate such attack through proactive proxy address shuffling. By shuffling proxy addresses, we can mutate the assignment between malicious

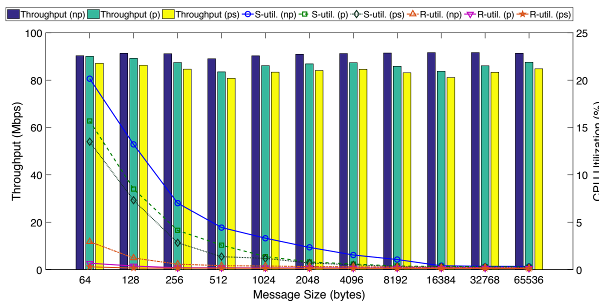


(a) Throughput of transferring files of varied sizes in three cases.

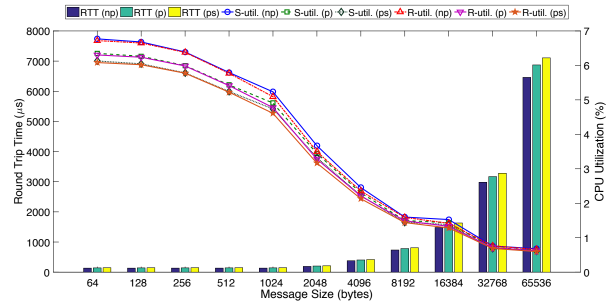


(b) Time for transferring a 5GB file with different shuffling intervals.

Fig. 5: SCP measurement results.



(a) Throughput overhead.



(b) Latency overhead.

Fig. 6: Netperf measurement results. S-util. represents sender's CPU utilization, R-util. represents receiver's CPU utilization.

clients and the proxies, and diffuse the flooding traffic to evacuate the overwhelmed proxy.

VIII. Related Work

Deception using honeypots. Honeypots are information system resources purposefully deployed for attack attraction, detection, and information gathering. Traditional honeypots employ virtualization to trap and monitor attackers [7]. By using the hypervisors, honeypots can be integrated into the network infrastructure and monitor attacker activities even if the system is compromised [9], [11]. Recently, there are increasing interests on using honeypots for developing countermeasure mechanisms [9], [10], [11]. However, the effectiveness of these honeypot-based defensive mechanisms has been largely constrained by either the low fidelity of the decoys or the poor scalability of decoy deployment platform. Though *honeyfarms* have been used for large-scale defense, they either require complicated scheduling schemes for dynamic resource allocation or cannot handle concurrent service requests towards a large number of network addresses [12], [13]. In contrast, our design can simultaneously present large numbers of high-fidelity decoy servers to the attackers with constrained system resources.

Microkernels, minimalistic OSes and unikernels. Microkernels or minimalistic OSes [27], [28] aim to provide just the required functionality for an application. In general, a microkernel executing at the privileged level exports the near-minimum set of mechanisms. Traditional OS functions, such as network protocol stacks, file systems and device drivers, are removed from the microkernel and are instead run in the user mode. However, most microkernels cannot run in virtualized environments. Another candidate for building the lightweight proxy platform incorporates unikernels, which are specialized, single address space LibOS-style virtual machines [17]. They are built by compiling high-level languages directly into specialized images running on a hypervisor. Examples of unikernels include ClickOS [16] and Mirage [29]. The design options of unikernels reduce the amount of deployed code, thus reducing the attack surface. Moreover, the small footprint of unikernel renders it competitive for deploying decoys in high density.

Network address randomization. Multiple techniques have been proposed to dynamically mutate the network attack surface [30] for anti-reconnaissance, including IP and MAC addresses, open ports, and network topology [31], [32], [33]. We also integrate IP and MAC address randomization into our framework to invalidate the attacker's scanning efforts. However, existing active connections may be disrupted during the randomization process, since the TCP layer protocols depend on a stationary IP address. Jafarin et al. propose to add an extra layer of mutable virtual IP addresses that are visible to the public world and conceal the real IP addresses from the attackers [32]. Similarly, Sun et al. use a dedicated module within both connection endpoints to intercept a connection from the network layer and translate the application-perceived IP address to the NIC-associated address [33]. However, we cannot directly adopt their strategies since our lightweight proxy's OS is truncated, lacking the necessary

mechanisms to support connection translation. We also do not want to complicate the deployment with an extra layer of virtual addresses. Instead we develop our own scheme to transparently migrate existing connections related to the proxies with changed addresses.

IX. CONCLUSIONS

We propose a decoy-based defensive framework that can proactively protect critical servers against targeted attacks. We design a highly scalable system that seamlessly integrates a large number of high-fidelity decoy servers with the critical real server infrastructures. By combining dynamic proxy address shuffling with transparent connection migration, our system can not only invalidate the attackers' scanning efforts, but also maintain the legitimate users' service connections. Our evaluation on a system prototype demonstrates the effectiveness of our approach in defeating targeted attacks and shows our system introduces small performance overhead.

X. ACKNOWLEDGMENT

Jianhua Sun and Kun Sun's work is supported by U.S. Office of Naval Research under grants N00014-16-1-3214 and N00014-16-1-3216. Qi Li's work is supported by NSFC under grant 61572278.

REFERENCES

[1] "2016 Internet Security Threat Report," https://www.symantec.com/security-center/threat-report.

[2] "Decoys and the Security of Deception," https://www.alluresecurity.com/blog/use-your-illusion-decoys-the-security-of-deception/.

[3] "Hacking Team and Defense through Deception," https://securityledger.com/2016/05/opinion-hacking-team-and-defense-through-deception/.

[4] "IBM Cost of Data Breach Study," http://www-03.ibm.com/security/data-breach/index.html.

[5] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Security and Privacy in Communication Networks - 5th International ICST Conference, SecureComm 2009, Athens, Greece, September 14-18, 2009, Revised Selected Papers*, 2009, pp. 51–70.

[6] J. Voris, J. Jermyn, N. Boggs, and S. Stolfo, "Fox in the trap: Thwarting masqueraders via automated decoy document deployment," in *Proceedings of the Eighth European Workshop on System Security*, ser. EuroSec '15, 2015.

[7] N. Provos and T. Holz, *Virtual Honeypots - From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 2008.

[8] L. Spitzner, "The honeynet project: Trapping the hackers," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 15–23, 2003.

[9] M. Beham, M. Vlad, and H. P. Reiser, "Intrusion detection and honeypots in nested virtualization environments," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[10] S. Kulkarni, M. Mutalik, P. Kulkarni, and T. Gupta, "Honeydoop - a system for on-demand virtual high interaction honeypots," in *Internet Technology And Secured Transactions, International Conference for*, 2012.

[11] T. K. Lengyel, J. Neumann, S. Maresca, B. D. Payne, and A. Kiayias, "Virtual machine introspection in a hybrid honeypot architecture," in *5th Workshop on Cyber Security Experimentation and Test, CSET '12, Bellevue, WA, USA, August 6, 2012*, 2012.

[12] X. Jiang, D. Xu, and Y. Wang, "Collapsar: A vm-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention," *J. Parallel Distrib. Comput.*, vol. 66, no. 9, pp. 1165–1180, 2006.

[13] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, fidelity, and containment in the potemkin virtual honeyfarm," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005*, 2005, pp. 148–162.

[14] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, "On recognizing virtual honeypots and countermeasures," in *Second International Symposium on Dependable Autonomic and Secure Computing (DASC 2006), 29 September - 1 October 2006, Indianapolis, Indiana, USA*, 2006, pp. 211–218.

[15] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005, pp. 29–36.

[16] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014.

[17] A. Madhavapeddy and D. J. Scott, "Unikernels: the rise of the virtual library operating system," *Commun. ACM*, vol. 57, no. 1, pp. 61–69, 2014.

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03, 2003.

[19] "Ryu," https://osrg.github.io/ryu/.

[20] "Snort," https://www.snort.org/.

[21] "Nicira Extension Structures," http://ryu.readthedocs.io/en/latest/nicira_ext_ref.html.

[22] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[23] "OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06)," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf.

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, Aug. 2000.

[25] "ClickOS Control," https://github.com/cnplab/clickos-ctl.

[26] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[27] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Minix 3: A highly reliable, self-repairing operating system," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 3, Jul. 2006.

[28] A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and performance in the denali isolation kernel," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, Dec. 2002.

[29] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13, 2013.

[30] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, 2011.

[31] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense - Creating Asymmetric Uncertainty for Cyber Threats*, ser. Advances in Information Security. Springer, 2011, vol. 54.

[32] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware IP address randomization for proactive agility against sophisticated attackers," in *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, 2015, pp. 738–746.

[33] J. Sun and K. Sun, "Desir: decoy enhanced seamless IP randomization," in *2016 IEEE Conference on Computer Communications, INFOCOM 2016, San Francisco, April 10 - April 15, 2016*, 2016.