Universal Honeyfarm Containment

Jay Chen[†], John McCullough, and Alex C. Snoeren

[†]New York University and University of California, San Diego
jchen@cs.nyu.edu {jmccullo, snoeren}@cs.ucsd.edu

Abstract. The growing sophistication of self-propagating worms and botnets presents a significant challenge for investigators to understand. While honeyfarms have emerged as a powerful tool for capturing and analyzing rapid malware, the size and complexity of large scale, high fidelity honeyfarms make them problematic to operate in a simultaneously safe and effective manner. This paper introduces a universe abstraction that guarantees isolation between multiple malware infestations in a single honeyfarm while maximizing the realism of the honeyfarm as observed by a propagating worm. We demonstrate that each malware strain can be completely isolated without distorting malware spreading behavior, and that this can in fact increase the scalability of honeyfarms.

1 Introduction

The ever-increasing reliance of people and critical infrastructures on the convenience and ubiquity of the Internet has caused an equally drastic rise in criminal activity designed to profit from this dependence. To capitalize on software vulnerabilities in the most effective manner, criminals use technology that gives them the ability to quickly and automatically compromise large numbers of Internet hosts. After the hosts have been compromised, they are used as a launching platform or bot-net for various types of often interconnected profit driven-schemes such as bulk-advertising email (SPAM), distributed denial-of-service (DDoS) extortion, identity theft (phishing), and software piracy [14]. This virtual ecology has evolved over recent years into a profit-driven conglomerate of organized crime.

A primary technology used to compromise hosts quickly and automatically is the Internet worm. The basic Internet worm is a piece of code that propagates itself through the Internet by infecting hosts and then using those compromised hosts to infect more hosts. The literature has identified numerous types of worm spreading behaviors based on simple random scanning, localized scanning, hit-list worms, and permutation scanning; there are also worms that infect through multiple vectors, slow-spreading worms, and flash worms [36]. One of the most powerful tools used in the study of and defense against malware is the honeypot [37]. A honeypot is simply a live system connected to the network that is carefully monitored to observe and study attacks [6]. An ideal honeypot machine behaves just as regular host does on the Internet, executing malicious code without restraint once compromised allowing researchers to observe zero-day or undiscovered exploits.

Researchers and commercial operators are collecting large numbers of honeypots together into systems called honeyfarms, which represent significant portions of the Internet address space. Various architectures have been designed across a spectrum of low-interaction/high-scale [31, 44] and high-interaction/low-scale honeyfarms [13, 38]. Along with our colleagues, we previously developed Potemkin, a high-interaction honeyfarm that can scale to thousands of end-hosts on a handful of physical machines without emulation to allow both fidelity and scalability [40]. Worms infecting Potemkin appear to have access to a normal Internet host running commodity operating system and application software.

The key concern with high-interaction honeyfarms like Potemkin is that the malware will exhibit its full behavior—including attempting to infect additional hosts. While observing this behavior is critical to epidemiological research and malware defense, the honeyfarm operator is legally and ethically obliged to avoid allowing the honeyfarm to infect hosts she does not own—the malware must be contained within the honeyfarm. Blocking out-bound traffic provides simple containment but prevents non-trivial infections. More complex containment policies are necessary to enable more sophisticated infection and control vectors. Current best practices include initially blocking connections to new hosts and steadily relaxing the policy as the behavior is better understood [7].

Unfortunately, the task of establishing an effective containment policy is confounded by the fact that a large honeyfarm is likely to contain many different contagions simultaneously. Different malware requires different containment policies and if two distinct pieces of malware infect the same honeypot, the different policies may not combine safely. Furthermore, the operator's attempts to study the malware in isolation will be frustrated. Current attempts to enforce the isolation of malware variants typically limit scalability and the ability to study the interaction of malware strains.

To address this issue, the Potemkin honeyfarm uses a new dynamic containment mechanism called a universe that can safely and automatically segregate each contagion, and allow them to spread dynamically throughout the honeyfarm. While described briefly in previous work, Potemkin's initial universe abstrction was quite simplistic and provided no ability for the honeyfarm operator to dynamically monitor or control the growth, interaction, or containment policies of individual universes. The contribution of this paper is to present the design and implementation of a full-fledged universe abstraction, as well as demonstrate its efficacy in the context of well-known, Windows-based malware strains.

Automatically and flexibly segregating each malware instance into its own universe benefits Internet epidemiology in at least two significant ways. First, we can increase the sample size of automatic malware signature generators in a non-trivial way. With existing technology, a fingerprint for a piece of malware can be quickly and automatically extracted from packet payloads [18, 34] or via host-level taint analysis [28] used by intrusion prevention systems [25, 41], but these techniques are quite sensitive to training data: they need to be provided with infected traffic that contains only one malware strain, and, in many cases, does not contain any benign traffic. Our universe abstraction ensures that the traffic contained in each universe is unpolluted by other malware strains. Second, by allowing the malware to spread inside its own universe, we can provide complete information about malware spreading behavior within a customizable population of operating systems and services. The community currently performs manual analysis on malware code to determine its intended behavior [22, 24, 33]; we allow the honeyfarm operator to quickly and automatically understand a worm's spreading behavior without laborious analysis of its code.

Beyond the immediate benefits of isolation, we show that the universe abstraction provides several major advantages:

- **Flexible containment policies.** Each universe has its own containment policy. Unlike previous honeyfarms that either opt for a strictly safe containment policy [13] or make potentially dangerous, general containment relaxations in order to gleaning more information from a particular piece of malware [7], we make it possible to enforce a relatively strong containment policy in general without giving up the ability to relax constraints on a relatively well-understood malware strain.
- **Controlled cross-contamination.** Existing honeyfarms rely on link-layer techniques (i.e., vLANs) to enforce strong, static separation between honey pots. By using a network-layer isolation technique, our approach allows for the dynamic provisioning of insulated regions of virtual honeyfarms. Furthermore, our network universe translation (NUT) mechanism allows for the controlled mingling of malware strains to support the study of symbiotic and parasitic contagions.
- **Safe service deployment.** By leveraging our flexible universe containment policies, we are able to deploy services or forwarding proxies within the honeyfarm without weakening containment policies. In particular, multiple universes can access the same, shared service without the ability to contact each other.

Universes introduce a great deal of flexibility to honeyfarm management, and determining the appropriate containment polices for any particular malware strain is beyond the scope of this paper. However, we show that even with simple policies the mechanisms we provide allow honeyfarms to obtain a higher degree of safety, realism, and scalability, and that these benefits can be easily leveraged for the purposes of Internet epidemiology. We begin in Section 2 with an overview of related work. In Section 3, we describe the basic architecture of Potemkin, our deployment framework. Section 4 describes the universe abstraction in detail, including network universe translation and service deployment. We briefly overview our Potemkin-based implementation in Section 5 before demonstrating the functionality of the deployed system in Section 6. Finally we conclude by discussing areas for future work in Section 7.

2 Related work

Researchers have developed a number of mechanisms to monitor Internet worms. One of the most significant early developments was the network telescope, which silently monitors large portions of un-allocated IP address space for scanning activity [26]. Telescopes proved especially effective for studying the prevalence of distributed denial-of-service attacks [23] and the spreading patterns of random-scanning worms such as Slammer [22] and CodeRed [24]. Over the years, several large-scale projects have emerged to collect and catalog attack data, including the SANS Internet Storm Center and Symantec's DeepSight system.

While useful, network telescopes are fundamentally limited by their inability to respond to in-bound traffic; telescopes can detect worm probing, but are unable to determine what further behavior a worm would exhibit. To address this limitation, lightweight active responders were developed that return responses to incoming probes. The simplest of these simply respond to TCP SYNs with SYN/ACKs in order to elicit the first part of the worm's payload [3]. Later work added protocol-specific responders to engage worms that required multiple protocol rounds [30, 44]. The attractiveness of these approaches is that they can be implemented in a stateless fashion, preserving the scalability of the telescope. The inability to keep state limits the complexity of the interactions that can be emulated, but by maintaining modest amounts of per-flow state, Provos significantly extended the emulation capabilities of active responders in his honeyd system [31]. Unfortunately, it too suffers from the inability to fully emulate the behavior of real operating systems and software. While more elaborate responders are being developed [2, 8], researchers interested in the behavior of worms in actual systems have no alternative other than to deploy live, infectable machines running real operating systems and applications. Such machines are known as honeypots [35]. Of course, attaching a real, physical machine to each IP address within a telescope is prohibitively expensive, so researchers have turned to virtual machine monitors (VMMs) such as VMWare, Xen, and user-mode Linux (UML) to support multiple honeypots on a single physical machine. These collections of honeypots have come to be known as honeynets [13] or honeyfarms, depending on their physical proximity.

A number of large-scale honeyfarms have been presented in the literature [7, 9, 16, 20, 40]. While each honeyfarm uses different techniques, they all must address the confinement problem: namely, ensure that worms cannot use honeypot machines to infect other machines on the Internet. Many of the existing systems such as Honeynet and Collapsar use intrusion detection systems (IDSs) like Snort-Inline [21] to block known attacks. Unfortunately, IDS-based methods do little to limit the spread of previously unknown worms. Hence, honeyfarms must implement additional containment mechanisms to prevent the spread of new malware variants. For example, GQ allows a limited number of out-bound connections per honeypot on well-known ports, HoneyStat uses a sophisticated causality tracking mechanism to determine when in-bound network events (i.e., worms) cause subsequent events, etc. To the best of our knowledge, however, no other honeyfarm supports multiple, simultaneous containment policies.

Similarly, honeyfarms control cross-infections by preventing worms in one honeypot from infecting a honeypot already infected by another variant. Rajab et al. addressed this problem in their VMWare-based honeynet [32] by statically allocating 128 distinct IP address to each honeypot and keeping each honeypot on a separate vLAN. They attempt to prevent a single VMM from hosting multiple simultaneous infections through a set of ad hoc heuristics, but there are no guarantees. In no instance can honeypots contact other, already-infected honeypots. In Virtual Playgrounds, Jiang et al. provide a disconnected emulation testbed for running worm propagation experiments for analysis [17]. While ensuring safety, disconnection from the Internet renders the system ineffective for studying real-time analysis. Also, given the complex nature of the experimental testbed being required for a particular experiment, it seems challenging to rapidly deploy a Virtual Playground to analyze 0-day worms with previously unknown signatures or infection vectors.

Perhaps the most similar honeyfarm system to Potemkin is the contemporaneously designed GQ system [7]. (Indeed, these similarities are not accidental as both are part of the same NSF-sponsored Collaborative Center for Internet Epidemiology and Defenses and are motivated by the same goals.) Their different design criteria informed very different approaches to isolation, however. GQ was designed around a commodity VMM-based honeyfarm and thus mandated the use of transparent network filtering and translation mechanisms that were backwards compatible with this existing infrastructure. Hence, GQ maps isolation groups onto vLAN tags exported to individual VMs. By contrast, the Potemkin system was designed around a custom low-cost VMM that targeted an environment with many thousands of simultaneously active honeypots. This environment allowed and encouraged us to create a ”pure” isolation abstraction; one that at once is extremely efficient, completely transparent to the network, and allows arbitrarily fine-grained control over the mixing of different infections in a honeyfarm.

3 Potemkin

While our universe concept is quite general, we provide a brief overview of our current honeyfarm platform, Potemkin [40], to facilitate a concrete discussion of our design. Potemkin aggressively multiplexes physical resources through extensive use of delta virtualization and flash cloning, achieving a two-to-three orders of magnitude improvement in scale when compared to bare-iron honeyfarms. One consequence of Potemkin's support for flash cloning is the ability to dynamically instantiate new honeypots, enabling the rapid and (relatively) unconstrained spread of malware within the honeyfarm. We begin by describing the purpose and functionality of the main components of Potemkin's architecture shown in Figure 1.

Potemkin gathers in-bound traffic through two mechanisms: routing and tunneling. Potemkin attracts traffic by configuring external Internet routers to tunnel packets destined for a particular address range back to the gateway. While

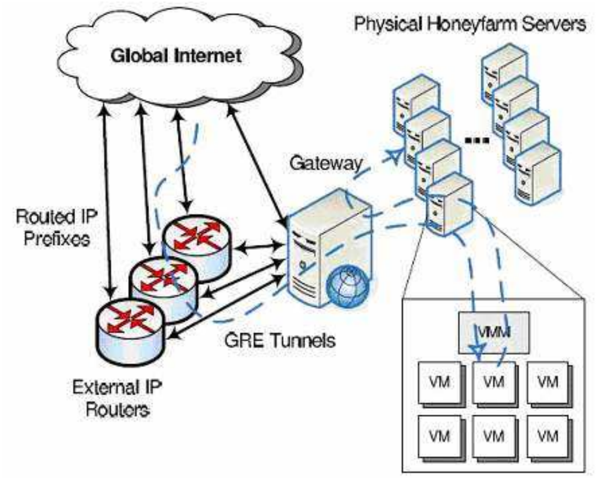


Fig. 1. The architecture of Potemkin [40, Fig. 1].

adding latency, tunneling is invisible to traceroute and can often be operationally and politically simpler for many organizations. These techniques are also used by several other honeyfarm designs such as Collapsar [16].

Potemkin does not maintain a physical machine for each IP address in the honeyfarm. Instead, it instantiates honeypot machines using a modified version of the Xen virtual machine monitor [4]. Modern Xen allows Potemkin to (in theory, at least) run any operating system, although our current deployment uses only Windows XP and Linux. To provide isolation, Potemkin creates a new virtual machine for each distinct honeypot machine. When a packet arrives for a new honeypot, the VMM (called a honeyfarm server) creates a new VM which is an identical copy of the reference image through a process called flash cloning. Once this new VM (hereafter referred to as a pot machine) boots, it adopts the packet's destination address and handles the request as though it were the intended recipient. Subsequent packets in the session are delivered directly to that pot machine. Since each honeypot is implemented by a distinct VM, any side effects from an attack will be isolated from other VMs. Finally, the VMM reclaims the resources of pot machines that are no longer active (or interesting to the honeyfarm operator) by destroying the cloned VM.

All traffic into and out of the honeyfarm is passed through a physical honeyfarm gateway. The principal role of the gateway is to provide containment and remove idleness in the IP address space utilization—in other words, to map the entire monitored address space down to the set of active flows communicating with currently instantiated honeypot machines. Packets that arrive at the gateway from the Internet are dynamically assigned to an appropriate back-end honeyfarm server. Packets destined for inactive IP addresses—those for which there is no current active pot machine—are sent to a honeyfarm server with sufficient resources to flash clone a new VM. The gateway also intelligently manages assignments of pot machines to in-bound traffic by offloading or discarding uninteresting behavior. In the case of network port scanning, instantiating a pot machine for each scan packet can inflate the demands on the honeyfarm unnecessarily. Instead, the gateway can decide to filter packets from the same source to multiple destinations (horizontal scans) or arrange for scans to be proxied (either by the gateway itself or a lightweight responder) and then migrate to a dedicated pot machine if a more substantive transaction occurs (in a fashion similar to RolePlayer [8]). Similarly, during a worm outbreak a large number of effectively identical infections will cross the gateway. Since a new identical infection is unlikely to impart additional knowledge, the gateway may use existing pattern-matching algorithms to quickly detect known attacks [18, 20, 34], and kill the universe rather than allowing it to grow. Finally, a universe will see any and all traffic a member pot machine generates, including service discovery traffic if that is turned on. In practice, this traffic is dropped, but adding filter rules at the gateway could allow it if desired. While crucial for effective honeyfarm operation, in-bound packet filtering rules are outside the scope of this paper; further details about the gateway's filtering mechanisms are available elsewhere [5, 39].

4 Universes

In Potemkin, as with most honeyfarms, pot machines are always allowed to communicate with machines that contact it (this is necessary, for example, to set up a TCP connection). This base-case rule, however, does not allow malware to spread, or, in the case of so-called phone-home or multi-stage worms that contain only an initial infection vector and need to download the rest of the malware from a central server. A common approach to enable worm infection is to allow compromised machines to communicate with the first Internet host they attempt to contact, under the assumption that this host is a central server or bot-master. Though this heuristic allows phone-home behavior, it could also allow the pot machine to infect an innocent host. Also, this policy would not permit propagation after the first host contacted. Hence, in Potemkin (and GQ), pot machines are allowed to attempt to contact any IP address they wish, but, unless the traffic is explicitly allowed out into the Internet (as in the case above of responding to an incoming connection), the connection is redirected back into the honeyfarm where a new VM assumes the IP address of the intended destination and masquerades as the Internet host. We call this process *reflection.*

Tracking these reflected communication patterns allow various behavioral detection mechanisms [7, 12, 43] to be leveraged to quickly gain insight into the spreading dynamics of new worms. Though, reflection appears to be a simple and viable solution, it introduces complications that cannot be solved in an elegant manner with existing mechanisms.

4.1 Preventing cross-contamination

With the infrastructure described so far, all traffic directed at a single IP address would be funneled to the pot machine that had assumed the identity of that IP

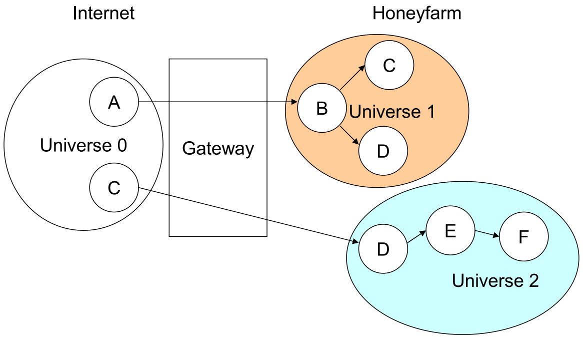


Fig. 2. Universal containment.

address inside the honeyfarm. Using the simplistic phone-home policy described above that allows pot machines to communicate with any Internet hosts that contact them, such a pot machine would be allowed to communicate with both Internet hosts, possibly leading to cross-contamination not only of the pot machine, but of the original Internet hosts as well. Incoming flows from different sources are directed to distinct pot machines, even if they are destined to the same IP address. The gateway creates flow mapping records to keep track of the unique source/destination destination pairs and facilitate packet forwarding.

To implement this functionality, we introduce the concept of a universe, an abstraction that provides the honeyfarm operator with a potentially infinite number of independent honeyfarm with which to analyze worm behavior. A universe represents a set of causally related pot machines that were created due to communication with each other. Upon creation, each pot machine in the honeyfarm is assigned a Universe Identifier (UID) depending on the flow that spawned the VM. If the flow originated from another pot machine, the new pot machine is assigned the same UID as the contacting pot. If, however, the flow originates outside the honeyfarm, a new universe is created with a unique UID.

All sources on the Internet are considered to have the reserved UID 0, and new universes are assigned monotonically increasing ID numbers. A packet destined for some address X not yet assigned in the honeyfarm creates a new universe U. Each subsequent pot machine created by communication rooted at X (i.e., either directly from X or any other pot machine in universe U) is also said to exist in universe U. The containment policy for a honeypot machine is dictated by its universe. In other words, if any machine in universe U is allowed to communicate with a given host, then all machines inside the universe are. The end result is that every causal chain of hosts initiated from a host on the Internet creates a new universe whose communication, both inside (to other pot machines with the same UID) and outside (Internet hosts or pot machines with other UIDs) of the universe is governed by a configurable, per-universe containment policy. In this way, we can allow infections to spread in the honeyfarm while being isolated from the Internet and each other.

Beyond elegantly solving our aliasing confusion, UIDs allow independent experimental test beds with which to analyze worm behavior, and also give us another metric by which we can measure and control the growth of each new universe spawned by an individual infection. In Figure 2, host A on the Internet (universe 0) infects host B in the honeyfarm, which is assigned to universe 1. Subsequently, B goes on to infect hosts C and D, which are all contained in universe 1. Concurrently, host C on the Internet is able to infect B, D, and E in universe 2.

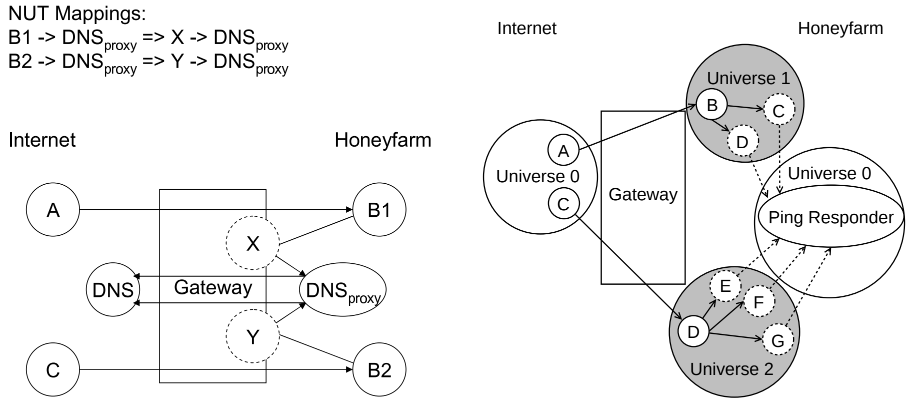
4.2 Network universe translation (NUT)

While complete isolation is usually desirable, there are occasions when it is convenient for pot machines from different universes to communicate, either with other universes inside the honeyfarm or with the external Internet. For example, several groups of symbiotic (e.g., Nimbda and Code-Red II) and parasitic (e.g., Bagel, Netsky, and MyDoom) worms have been discovered, where their complete spreading behavior is not exhibited unless they come into contact with hosts previously infected by another, specific piece of malware. Hence, a universe containment policy may specify that pot machines within the universe are allowed to communicate with specific other universes. While these universes are specified by UID, we expect such policy rules to be inserted dynamically by protocol analyzers that determine what malware strains exist inside each universe. We have not yet implemented such automation in Potemkin, however.

The challenge with allowing inter-universe communication is that an IP address may exist in both universes, either because the respective pot machines were created before the containment policy was relaxed, or because the containment policy discriminates between packets destined to the same IP address (e.g., allows TCP response packets out but reflects SYNs). We resolve this potential ambiguity through network universe translation (NUT). If a packet is destined for the IP address of a pot machine in another universe—and the source universe containment policy allows—the gateway creates a NUT mapping by generating a unique, random ephemeral IP address in the destination universe and rewriting the source address of the packet to appear to come from this random address within the destination universe. From the point of view of the destination pot machine, it receives a packet from a random source inside its own universe. If and when it responds, the packet is de-NUT'ed by the gateway to the proper universe and IP address by consulting the NUT mapping table.

4.3 Multiverses

In addition to allowing malware strains to mingle, NUT also conveniently addresses another operational hurdle to practical honeyfarm deployment. Worms frequently make use of standard Internet services such as DNS, and modern worms increasingly access well-known Internet sites. For example, Santy is a hit-list worm that identifies potential victims by querying Google [15]. While it is straightforward to adjust a universe's containment policy to allow outgoing



(a) DNS proxy ambiguity (packet path view)

(b) ICMP Ping Responder (logical view)

Fig. 3. Network Universe Translation

connections to these services, such a policy admits the possibility that the worm might try to attack or infect the service itself. It is generally preferable, then, to implement a service replica (or a transparent protocol-scrubbing proxy) within the honeyfarm. Rather than deploy multiple instances of the service—one in each universe—the single service instance can be made available in multiple universes simultaneously through NUT.

In our architecture, each service exists in its own universe with an appropriate containment policy. In general, service universes (also known as multiverses) will likely be allowed to contact any Internet host. So far we have implemented two services that are accessible from multiple universes: a DNS proxy and an ICMP Echo responder. Through NUT we are able to create the illusion of a service in each universe while using only a single physical (or virtual) resource instance. Thus, NUT allows the honeyfarm to multiplex resources and lower configuration and management overhead without modifying the server itself. Each universe can specify what types of packets are allowed to enter service universes. Currently, in the case of DNS, we configure all of the pot machines to use a well-known IP address for the local DNS resolver. In the case of ICMP echo requests we simply filter on the protocol fields and route all ICMP echo packets not destined to addresses already existing within a universe to the responder multiverse.

DNS proxy. The simplest approach to enabling DNS with a honeyfarm is to adjust the containment policy to allow pot machines to send out DNS requests directly to name servers on the Internet. Given the potential for malware to attack DNS servers, however, we chose to setup a local DNS caching proxy to forward the (scrubbed) requests. Conveniently, having a central caching server also allows us to reduce the DNS load of a spreading malware instance. Without NUT, such a configuration would lead to undefined behavior. For example, in Figure 3(a) (ignoring X and Y for now) there are two non-service universes created by hosts A and C that both contain a pot machine with IP address B, $B1$ and $B2$ respectively. DNS queries sent from $B1$ in universe 1 and $B2$ in universe 2 would be routed to the DNS proxy. However, when the responses return the DNS proxy would send them on to B, and the gateway would not know which universe's B they should be directed to. (The pot machines—and, thus, the applications therein—have no notion of universes; that state is kept entirely at the gateway.)

Using network universe translation, however, requests from hosts $B1$ and $B2$ go through the gateway and are NUT'ed to have source IP addresses X and Y, respectively. When responses return, the DNS proxy sends them to X and Y accordingly, and they are translated by the gateway back to $B1$ and $B2$ and sent to the appropriate hosts in the proper universe. In effect, NUT is similar to Network Address Translation (NAT), except nonce IP addresses are used to multiplex requests from multiple universes as opposed to nonce ports to multiplex requests from multiple IP addresses.

ICMP ping responder. ICMP ping packets are often used in reconnaissance to scan for the presence or absence of hosts in an IP address range. Pings are relatively innocuous in that they provide little information[1], but it would help preserve the illusion that the honeyfarm's monitored IP address space is in fact filled with active hosts if the ping packets are responded to. Hence, all ICMP requests should be met with an ICMP reply, but allowing ping packets to create potentially thousands of VMs a second would be a waste. Instead, by default ping packets are not allowed to create new identities; each universe sends all ping packets to a single ping responder multiverse unless the destination in question already exists. The gateway considers the ping responder as a "service" similar to DNS. A single pot machine handle all ICMP ping packets using NUT by making the single ping responder resource appear in all universes where it is required, as shown in Figure 3(b). Hosts C, D, E, F, and G are all represented by the single virtual machine "PingResponder." Pings directed to these identities are forwarded via NUT to the ping responder.

5 Implementation

While there are potentially many realizations of the universe abstraction, ours follows from the network architecture of Potemkin. In particular, the Potemkin gateway is equipped with two physical Ethernet interfaces that allow it to straddle two separate virtual LANs (vLANs). The external vLAN faces the Internet, and traffic passes through a next-hop router that lies between the gateway and the Internet. The physical honeypot machines are all connected to a single internal vLAN (although, if desired, each machine could be placed on its own vLAN [32]). ARP is not run on the honeyfarm vLAN. Instead, the MAC address of each honeyfarm server (VMM) is registered with the gateway on startup.

[1] It is possible to fingerprint a machine's operating system through ICMP [1]; hence, operators may wish to configure multiple responder multiverses, one per OS type.

When the gateway wants to forward a packet to a pot machine it uses the MAC address of the hosting honeyfarm server. In the reverse direction the gateway serves as the next-hop router for all pot VMs. This approach avoids the need for IP routing or packet rewriting inside the honeypot.

In order to avoid modifying the software of any honeypot VMs (we want to be able to run exact replicas of real systems, after all), the universe abstraction is implemented entirely within the gateway. There is no need for any affinity between universes and VMMs: a physical honeyfarm machine can host pot machines (VMs) from any universe. The key is to ensure that no VMM hosts two pot VMs with the same IP address (which must be in different universes). As long as IP addresses uniquely identify a particular pot machine on each physical honeyfarm machine, the gateway can simply forward packets to the appropriate VMM, who can demultiplex the packet based upon destination IP address, and neither the VMM or pot VMs need to know which universe is involved.

5.1 Gateway architecture

The Potemkin gateway is built on top of the Click modular software router framework [19], as shown in Figure 4. A router implemented in Click consists of a set of packet-processing modules called elements, implemented as C++ classes. Using a special description language (a Click "configuration"), elements can be connected together to form a directed graph that represents how packets flow through the processing modules. Our implementation adds roughly 7,800 lines of custom element code to the base Click 1.4.3 installation, in addition to 450 lines of configuration and 1,000 lines for an administrative interface. The Click elements are grouped generally into five separate stages: packet validation, packet filtering, packet rewriting, packet encapsulation, and logging. The in-bound and out-bound packet paths are similar in that they both go through the same stages, but the actual elements and logic are sometimes quite different.

1. **Packet validation.** The packet validation stage in-bound from the Internet consists of a series of simple Click elements to ensure that only packets from the Internet destined to the honeyfarm's monitored IP address space are allowed into the honeyfarm. The out-bound validation stage ensures that packets are properly formed and do not exceed the MTU of the data-link layer of the next hop when exiting the honeyfarm.
2. **Packet filtering.** The primary purpose of the packet filtering stage is to limit the resources required to support the honeyfarm. Currently, filters in our running configuration include a simple blacklist, scan filter, and a universe freeze filter. The scan filter is designed to rate limit horizontal scans that target the same port on several hosts, a common behavior of scanning worms [42].
3. **Packet redirection & rewriting.** This is where the gateway makes forwarding and redirection decisions according to each universe's containment policy. Rewriting is only necessary in NUT and a few other rare cases described in the following section.

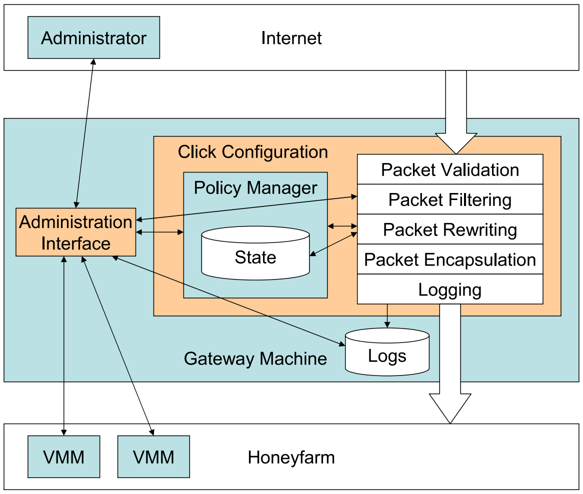


Fig. 4. The Potemkin network gateway architecture

4. **Packet encapsulation.** In this stage, the IP packets go through a final sanity check and are prepared to be delivered to their final destination. Packets destined into the honeyfarm are encapsulated with an Ethernet header containing the destination MAC address of the physical machine that will be responsible for instantiating a VM to represent the destination IP address of the packet. The packet is then transmitted onto the honeyfarm's internal VLAN and picked up by the VMM of the machine it is destined to. The IP packets that are intended for the Internet are checksummed one final time, and GRE encapsulated with the destination IP address of the tunnel endpoint.
5. **Logging.** All packets received and delivered by the gateway on either interface are logged to a file in a tcpdump-like format.

The gateway maintains five distinct classes of global state: types, identities, machines, flows, and histories. Type entries store the configuration details of potential pot machines. Identities track the state of instantiated pot machines. Machine entries track the physical honeyfarm server resources available to the gateway for allocation. Finally, flow and history entries store information about communication between pot machines in the honeyfarm and hosts on the Internet or other pot machines.

As mentioned previously, the assignment of IP addresses to pot machine images (i.e., OS version) in the honeyfarm can be made randomly or can be biased to present the illusion that a given IP address range hosts a particular distribution of software configurations. For example the operator could specify that some /16 address range contains 75% Windows machines and 25% Linux machines with different disk images.

Similar to a network address translator, the gateway maintains a flow-mapping data structure for each flow in the honeyfarm that maps the flow from an incoming hardware interface and MAC address (i.e., honeyfarm machine or external router) to outgoing address and interface. When a packet for a new flow arrives at the gateway, it consults the appropriate universe's containment policy to decide how to forward it. Assuming the packet is not dropped, a new flow map is created and the gateway records a history entry with the time the flow was created, and by whom. An identity entry is instantiated for the destination IP address and its type and honeyfarm server machine are determined by the universe policy settings. To keep the amount of active state manageable, flow maps are cached for quick access and removed when the TCP session ends or after a default 5-minute timeout leaving only the corresponding history state.

If a packet arrives at the gateway belonging to a garbage-collected flow (i.e., the flow-map entry was purged from the cache) then the proper mapping can be reconstructed from the history state and the flow mapping entry re-inserted into the flow-map cache. This is useful since flow mappings are more than twice as large as history entries since long-term state can be maintained in a more compact but less efficient to access form. If history were not maintained, after the garbage-collection threshold the containment policy may disallow (or redirect) a flow that was previously established.

5.2 Network address translation (NAT)

Despite the flat Ethernet structure of the honeyfarm itself, it is occasionally necessary for the gateway to rewrite IP packets. When combining reflection with the universe abstraction, it is possible that a pot machine is allowed to send packets onto the Internet, but is using an IP address that does not physically belong to the honeyfarm address space. In particular, a pot machine created due to a reflected connection can have any IP address—most likely one outside the honeyfarm's address space. If that pot machine attempts to contact an Internet address allowed by the universe containment policy (such as the host that initially created the universe), response packets should be routed to the real Internet host, not the a masquerading pot machine. We address this issue by implementing network address translation (NAT) [11] at the gateway. Should a pot machine created by a reflected connection need to communicate with Internet hosts, we rewrite the source address to be inside the honeyfarm's address space so response packets will reach the gateway. We extend the identity entries, which keep track of the IP addresses being represented in the honeyfarm, with an alias field to keep track of the IP address a host is NAT'ed to.

5.3 Spoofing and backscatter

An often-used attack vector for malware is source IP-spoofing, where the source IP address of packets is forged. Given the infrastructure that we have described so far, it is interesting to examine what occurs when a host decides to send spoofed packets. If spoofed packets enter the honeyfarm from the Internet, the pot machine may generate response packets that create additional pot machines for the forged IP address—a phenomenon known as backscatter [27].

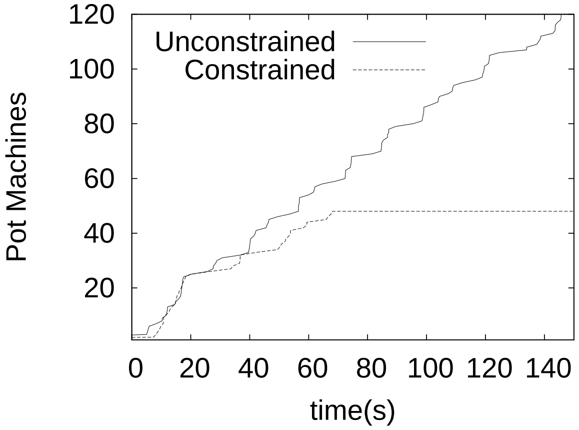


Fig. 5. Rate limiting and universe expansion caps.

Machines instantiated from in-bound backscatter packets are unlikely to respond in any useful way, therefore universe containment policies by default simply prevent instantiating pot machines for response packets (e.g., non-SYN TCP segments).

Unfortunately, if spoofed packets are sent from inside the honeyfarm, the default behavior is not as realistic. In general, the gateway will be unable to resolve the source IP/Ethernet address combination to a valid source universe. The only safe response in such a situation is to log the event and drop the packet. However, if, by chance, the source IP/Ethernet pair does exist, it would be possible for a host on the honeyfarm to send spoofed packets across universes. Hence, spoofing causes two problems: dropping spoofed packets is unrealistic because packet spoofing is common bot-net behavior, but potentially allowing cross universe packets is clearly a violation of the containment requirement.

To avoid this situation the VMMs on each honeyfarm server ensure that packets sent from hosted pot machines do indeed have the proper source IP address of the VM that they are being sent from. If the VMMs discover that the source IP address is being spoofed, they simply add a shim header between the Ethernet and IP headers to notify the gateway of the actual IP address of the sender. Upon receipt, the gateway looks up the source universe according to the actual IP address of the sender, and ensures that the packet is delivered to a pot machine within the same universe (or dropped, depending on the containment policy) so isolation not violated by spoofing.

6 Evaluation

Our universe abstraction is currently deployed within Potemkin on the UC San Diego campus. We have designed and implemented an extensive test suite to ensure the proper operation of the honeyfarm gateway and its containment policies; space constraints prevent us from including them here, but full details are

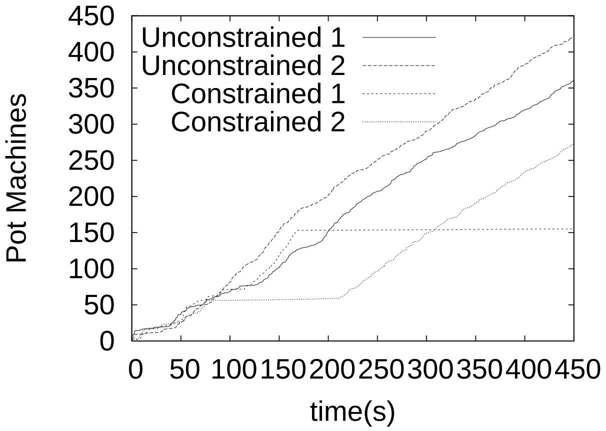


Fig. 6. Two distinct containment policies.

available elsewhere [5]. Similarly, extensive benchmarking of the gateway including forwarding speed and memory usage are discussed in another document [39]. Here, we provide a brief demonstration of the universe functionality by presenting the isolation and control capabilities of the gateway.

To simplify the discussion, we show the Potemkin honeyfarm being infected by Blaster, an older scanning worm. Blaster uses a buffer overflow in the Windows RPC service (on TCP port 135) to open a remote shell on TCP port 4444. The attacker then contacts the victim on port 4444 and instructs it to downloads and executes the binary from the infecting host via TFTP. New hosts are infected by sequentially scanning randomly selected subnets. In order to spread within the honeyfarm, the containment policy must allow multiple connections from the attacker to the victim (to allow the issuance of the TFTP command), and out-bound UDP connections from the victim back to the attacker (to allow the TFTP download itself). Blaster also has the ability to launch a DDoS attack against a Microsoft Web site, but that behavior is time-sensitive and not activated. We use Blaster's predictable scanning behavior to demonstrate the containment properties of the gateway.

In the scenario depicted in Figure 5, we demonstrate our ability to constrain the growth of the worm within the honeyfarm by applying various constraints on the infected universe. Both lines in the graph represent the total number of pot machines created by the Blaster infection. The number of pot machines created is not directly related to the number of hosts infected by Blaster, however, as the worm's implementation is fragile: an attempted infection attempt can be interrupted by a subsequent probe response, the TFTP service is not always left open, and the worm has a high probability of crashing individual hosts. For our current purposes, we plot only the number of pot machines scanned, not necessarily infected or scanning themselves.

The "unconstrained" line shows the rate of expansion of a universe initiated by an infection from an outside source. In order to prevent a rapid, exponential growth in the number of pot machines, the universe containment policy throttles scanning to a constant rate. Note the rate is applied to all the pot machines within the universe in aggregate, not individually (otherwise the aggregate growth would be exponential). In the second, constrained case, we demonstrate our ability to cap universe growth while allowing the existing machines to persist. We initiate the infection identically, with a similar initial containment policy, but at 70 seconds we constrict growth by preventing further universe expansion. Note that we are not preventing infected hosts from scanning, or existing hosts from becoming infected; we are only preventing the creation of further pot machines for this universe. This capability is critical to control the resource utilization of a new malware variant until its scanning behavior is understood—otherwise the entire honeyfarm could be swamped in a matter of minutes.

In our second scenario (Figure 6), we illustrate two simultaneous infections in the honeyfarm. Two separate outside sources initiate Blaster infections to the same IP address. Through our isolation mechanism, the outside sources infect a distinct pot machines in separate universes. In the unconstrained case, the two infections are allowed to grow at linear rates as in Figure 5. While the subtle details of worm timing and resource constraints prevent identical growth, both "Unconstrained 1" and "Unconstrained 2" ultimately achieve approximately similar growth rates. In the constrained case, we seek to show how the honeyfarm operator might vary their focus and resource allocations over time. Two infections are created identically to the unconstrained case; at 60 seconds we constrict the growth of "Constrained 2", choosing to allow "Constrained 1" to continue to grow. At 170 seconds, we also constrict the growth of "Constrained 1" and, at 220 seconds we permit further spreading of "Constrained 2." Each stage was implemented by dynamically adjusting the respective universe containment policy. Note that the machines within the universes may have identical IP addresses, so simple packet-based containment policies would be unable to produce this behavior. Only through our universe mechanism are we able to selectively control the growth distinct infections that remain isolated from one another.

7 Conclusion

We have presented the design and implementation of a universe abstraction that enables dynamic containment of malware inside a honeyfarm. Critically, universes allow controlled interaction of distinct malware strains, as well as the safe sharing of Internet services across honeypots. By implementing universes in the Potemkin gateway, we have demonstrated that universes are not only a useful mechanism, but a powerful abstraction for Internet epidemiology in general. By providing robust containment without the need for per-VM vLANs, our universe mechanism enables the deployment of highly dynamic, large-scale honeyfarms. Furthermore, no changes need to be made to the OS or application software on the honeypot machines themselves, providing complete realism.

With the universe mechanism in place, we are now considering a variety of plug-in technologies automatically adjust containment policies and detect malware strains, both at the network level [18, 34] and at the VM level [10, 29]. Worm detectors and taint checkers both suffer from false-positives and false-negatives, and the use of multiple universes provides an opportunity to generate additional samples with which to generate signatures. In particular, the universe abstraction allows us to provide clean training data where we can be assured all traffic in a universe is from a particular malware variant.

Finally, while still the preeminent method of detecting and collecting Internet worms, telescope-based honeyfarms are becoming increasingly less useful as malware begins to spread through other, non-scanning means such as email and messenger clients. Fortunately, our containment mechanisms allow us to use Potemkin not only as a honeyfarm, but as a closed environment in which to study malware obtained from a variety of channels, most notably SPAM. By using NUT to deploy needed services and application gateways inside of a multiverse, we are able to safely and effectively execute known bot-net and spyware code without the possibility of it leaking out onto the Internet.

Acknowledgments

The authors wish to thank Erik VandeKieft, who implemented the original Potemkin gateway, and Michael Vrable who developed the original Potemkin VMM and provided a great deal of on-going assistance. We are indebted to Stefan Savage, Geoff Voelker, and all the members of the Collaborative Center for Internet Epidemiology and Defenses (CCIED) for their thoughtful discussions and comments on earlier drafts of this paper. This work was supported by the National Science Foundation through a Cybertrust grant (CNS-0433668).

References

1. Remote OS detection via TCP/IP fingerprinting (2nd generation). `http://insecure.org/nmap/osdetect/`.
2. P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The Nepenthes platform: An efficient approach to collect malware. In Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID), Sept. 2006.
3. M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS '05), San Diego, CA, Feb. 2005.
4. 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 19th ACM Symposium on Operating System Principles (SOSP), Bolton Landing, NY, Oct. 2003.
5. J. Chen. Safe and realistic containment mechanisms for large-scale virtual honeyfarms. Master's thesis, UC San Diego, Sept. 2006.
6. B. Cheswick. An Evening with Berferd In Which a Cracker is Lured, Endured, and Studied. In Proceedings of the Winter Usenix Conference, San Francisco, CA, 1992.

7. W. Cui, V. Paxson, and N. Weaver. GQ: Realizing a System to Catch Worms in a Quarter Million Places. Technical Report TR-06-004, ICSI, Sept. 2006.
8. W. Cui, V. Paxson, N. Weaver, and R. H. Katz. Protocol-independent adaptive replay of application dialog. In Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS '06), San Diego, CA, Feb. 2006.
9. D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honeypots. In In Recent Advances In Intrusion Detection (RAID) 2004, Sept. 2004.
10. G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In Proceedings of the 5th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI), Boston, MA, Dec. 2002.
11. K. Egevang and P. Francis. RFC 1631 - The IP Network Address Translator (NAT). RFC 1631, May 1994.
12. D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A Behavioral Approach to Worm Detection. In Proceedings of the ACM Workshop on Rapid Malcode (WORM), Fairfax, VA, Oct. 2004.
13. Honeynet Project. Know Your Enemy: Learning about Security Threats. Pearson Education, Inc., Boston, MA, second edition, 2004.
14. Honeynet Project. Know Your Enemy: Tracking Botnets. http://www.honeynet.org/papers/bots/, Mar. 2005.
15. Internet Storm Center. Santy worm.
16. X. Jiang and D. Xu. Collapsar: A VM-Based Architecture for Network Attack Detention Center. In Proceedings of the USENIX Security Symposium, San Diego, CA, Aug. 2004.
17. X. Jiang, D. Xu, H. J. Wang, and E. H. Spafford. Virtual playgrounds for worm behavior investigation. In Proceedings of Symposium on Recent Advances in Intrusion Detection (RAID), Sept. 2005.
18. H.-A. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In Proceedings of the USENIX Security Symposium, San Diego, CA, Aug. 2004.
19. E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. ACM Transactions on Computer Systems, 18(3):263–297, Aug. 2000.
20. C. Kreibich and J. Crowcroft. Honeycomb — Creating Intrusion Detection Signatures Using Honeypots. In Proceedings of the 2nd ACM Workshop on Hot Topics in Networks (HotNets-II), Cambridge, MA, Nov. 2003.
21. W. Metcalf. Snort inline. `http://snort-inline.sourceforge.net`.
22. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. IEEE Security and Privacy, 1(4):33–39, July 2003.
23. D. Moore, C. Shannon, D. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. ACM Transactions on Computer Systems, 24(2):115–139, May 2006.
24. D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In Proceedings of the ACM/USENIX Internet Measurement Workshop (IMW), Marseille, France, Nov. 2002.
25. D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In Proceedings of the IEEE Infocom Conference, San Francisco, California, Apr. 2003.
26. D. Moore, C. Shannon, G. Voelker, and S. Savage. Network telescopes: Technical report. Technical Report CS2004-0795, UCSD, July 2004.

27. D. Moore, G. M. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In Proceedings of the USENIX Security Symposium, Washington, D.C., Aug. 2001.
28. J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS '05), San Diego, CA, Feb. 2005.
29. J. Newsome and D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS '05), San Diego, CA, Feb. 2005.
30. R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In Proceedings of the USENIX/ACM Internet Measurement Conference, Taormina, Sicily, Italy, Oct. 2004.
31. N. Provos. A Virtual Honeypot Framework. In Proceedings of the USENIX Security Symposium, San Diego, CA, Aug. 2004.
32. M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In Proceedings of the USENIX/ACM Internet Measurement Conference, pages 41–52, Rio de Janeiro, Brazil, Oct. 2006.
33. C. Shannon and D. Moore. The spread of the witty worm. IEEE Security and Privacy, 2(4), July 2004.
34. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. In Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI), San Francisco, CA, Dec. 2004.
35. L. Spitzner. Honeypots: Tracking Hackers. Addison Wesley, 2003.
36. S. Staniford, V. Paxson, and N. Weaver. How to 0wn the internet in your spare time. In Proceedings of the USENIX Security Symposium, San Francisco, CA, Aug. 2002.
37. C. Stoll. The Cuckoo's Egg. Pocket Books, New York, NY, 1990.
38. Symantec. Decoy Server Product Sheet. http://www.symantec.com/.
39. E. Vandekieft. Network Address Translation for Honeypot Farms. Master's thesis, UC San Diego, Dec. 2004.
40. 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 System Principles (SOSP), Brighton, UK, Oct. 2005.
41. H. Wang, C. Guo, D. Simon, and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In Proceedings of the ACM SIGCOMM Conference, Portland, Oregon, Sept. 2004.
42. N. Weaver, S. Staniford, and V. Paxson. Very Fast Containment of Scanning Worms. In Proceedings of the USENIX Security Symposium, San Diego, CA, Aug. 2004.
43. J. Xiong. ACT: Attachment Chain Tracing Scheme for Email Virus Detection and Control. In Proceedings of the ACM Workshop on Rapid Malcode (WORM), Fairfax, VA, Oct. 2004.
44. V. Yegneswaran, P. Barford, and D. Plonka. On the Design and Use of Internet Sinks for Network Abuse Monitoring. In Proceedings of Symposium on Recent Advances in Intrusion Detection (RAID), Sept. 2004.