1

Composite Hybrid Techniques For Defending Against Targeted Attacks

Stelios Sidiroglou and Angelos D. Keromytis

Department of Computer Science, Columbia University
1214 Amsterdam Ave. New York, NY 10027, USA
{stelios,angelos}@cs.columbia.edu

Summary. We investigate the use of hybrid techniques as a defensive mechanism against targeted attacks and introduce *Shadow Honeypots*, a novel hybrid architecture that combines the best features of honeypots and anomaly detection. At a high level, we use a variety of anomaly detectors to monitor all traffic to a protected network/service. Traffic that is considered anomalous is processed by a "shadow honeypot" to determine the accuracy of the anomaly prediction. The shadow is an instance of the protected software that shares all internal state with a regular ("production") instance of the application, and is instrumented to detect potential attacks. Attacks against the shadow are caught, and any incurred state changes are discarded. Legitimate traffic that was misclassified will be validated by the shadow and will be handled correctly by the system transparently to the end user. The outcome of processing a request by the shadow is used to filter future attack instances and could be used to update the anomaly detector.

Our architecture allows system designers to fine-tune systems for performance, since false positives will be filtered by the shadow. Contrary to regular honeypots, our architecture can be used both for server and client applications. We also explore the notion of using Shadow Honeypots in Application Communities in order to amortize the cost of instrumentation and detection across a number of autonomous hosts.

1.1 Introduction

Due to the increasing level of malicious activity seen on today's Internet, organizations are beginning to deploy mechanisms for detecting and responding to new attacks or suspicious activity, called Intrusion Prevention Systems (IPS). Since current IPS's use rule-based intrusion detection systems (IDS) such as Snort [37] to detect attacks, they are limited to protecting, for the most part, against already known attacks. As a result, new detection mechanisms are being developed for use in more powerful reactive-defense systems. The two primary such mechanisms are honeypots [32, 13, 62, 45, 20, 6] and anomaly detection systems (ADS) [53, 57, 52, 8, 19]. In contrast with IDS's, honeypots and ADS's offer the possibility of detecting (and thus responding to) previously unknown attacks, also referred to as *zero-day attacks*.

Honeypots and anomaly detection systems offer different tradeoffs between accuracy and scope of attacks that can be detected, as shown in Figure 1.1. Honeypots can be heavily instrumented to accurately detect attacks, but depend on an attacker attempting to exploit a vulnerability against them. This makes them good for detecting scanning worms [2, 3, 13], but ineffective against manual directed attacks or topological and hit-list worms [48, 47]. Furthermore, honeypots can typically only be used for server-type applications. Anomaly detection systems can theoretically detect both types of attacks, but are usually much less accurate. Most such systems offer a tradeoff between false positive (FP) and false negative (FN) rates. For example, it is often possible to tune the system to detect more *potential* attacks, at an increased risk of *misclassifying* legitimate traffic (low FN, high FP); alternatively, it is possible to make an anomaly detection system more insensitive to attacks, at the risk of missing some real attacks (high FN, low FP). Because an ADS-based IPS can adversely affect legitimate traffic (*e.g.,* drop a legitimate request), system designers often tune the system for low false positive rates, potentially misclassifying attacks as legitimate traffic.

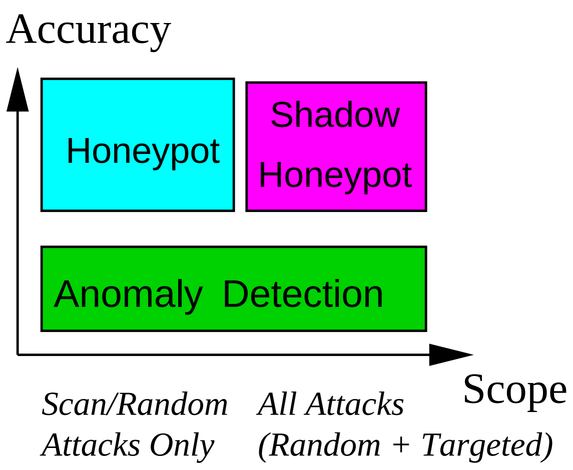

Fig. 1.1. A simple classification of honeypots and anomaly detection systems, based on attack detection accuracy and scope of detected attacks. Targeted attacks may use lists of known (potentially) vulnerable servers, while scan-based attacks will target any system that is believed to run a vulnerable service. AD systems can detect both types of attacks, but with lower accuracy than a specially instrumented system (honeypot). However, honeypots are blind to targeted attacks, and may not see a scanning attack until after it has succeeded against the real server.

We propose a novel hybrid approach that combines the best features of honeypots and anomaly detection, named *Shadow Honeypots*. At a high level, we use a variety of anomaly detectors to monitor all traffic to a protected network. Traffic that is considered anomalous is processed by a shadow honeypot. The shadow version is an instance of the protected application (*e.g.,* a web server or client) that shares all internal state with a “normal” instance of the application, but is instrumented to detect potential attacks. Attacks against the shadow honeypot are caught and any incurred state changes are discarded. Legitimate traffic that was misclassified by the anomaly detector will be validated by the shadow honeypot and will be *transparently* handled correctly by the system (*i.e.,* an HTTP request that was mistakenly flagged as suspicious will be served correctly). Our approach offers several advantages over stand-alone ADS’s or honeypots:

- First, it allows system designers to tune the anomaly detection system for low false negative rates, minimizing the risk of misclassifying a real attack as legitimate traffic, since any false positives will be weeded out by the shadow honeypot.
- Second, and in contrast to typical honeypots, our approach can defend against attacks that are *tailored* against a specific site with a particular internal state. Honeypots may be blind to such attacks, since they are not typically mirror images of the protected application.
- Third, shadow honeypots can also be instantiated in a form that is particularly well-suited for protecting against *client-side* attacks, such as those directed against web browsers and P2P file sharing clients.
- Finally, our system architecture facilitates easy integration of additional detection mechanisms.

In addition to the server-side scenario, we also investigate a client-targeting attack-detection scenario, unique to shadow honeypots, where we apply the detection heuristics to content retrieved by protected clients and feed any positives to shadow honeypots for further analysis. Unlike traditional honeypots, which are idle whilst waiting for active attackers to probe them, this scenario enables the detection of passive attacks, where the attacker lures a victim user to download malicious data.

Finally, we explore the combination of Shadow Honeypots with Application Communities to create a distributed collaborative environment where detection and the processing cost incured by the use Shadow Honeypots is shared across a large number of hosts.

Paper Organization

The remainder of this paper is organized as follows. Section 1.2 discusses the shadow honeypot architecture in greater detail. Some of the limitations of our approach are briefly discussed in Section 1.3. We give an overview of related work in Section 1.4, and conclude the paper with a summary of our work and plans for future work in Section 1.5.

1.2 Architecture

The Shadow Honeypot architecture is a systems approach to handling network-based attacks, combining filtering, anomaly detection systems and honeypots in a way that exploits the best features of these mechanisms, while shielding their limitations. We focus on transactional applications, *i.e.,* those that handle a series of discrete requests. Our architecture is *not* limited to server applications, but can be used for client-side applications such as web browsers, P2P clients, *etc.* As illustrated in Figure 1.2, the architecture is composed of three main components: a filtering engine, an array of anomaly detection sensors and the shadow honeypot, which validates the predictions of the anomaly detectors. The processing logic of the system is shown graphically in Figure 1.3.

The filtering component blocks known attacks. Such filtering is done based either on payload content [56, 1] or on the source of the attack, if it can be identified with reasonable confidence (*e.g.,* confirmed traffic bi-directionality). Effectively, the filtering component short-circuits the detection heuristics or shadow testing results by immediately dropping specific types of requests before any further processing is done.

Traffic passing the first stage is processed by one or more anomaly detectors. There are several types of anomaly detectors that may be used in our system, including payload analysis [57, 42, 18, 52] and network behavior [16, 60]. Although we do not impose any particular requirements on the AD component of our system, it is preferable to tune such detectors towards high sensitivity (at the cost of increased false positives). The anomaly detectors, in turn, signal to the protected application whether a request is potentially dangerous.

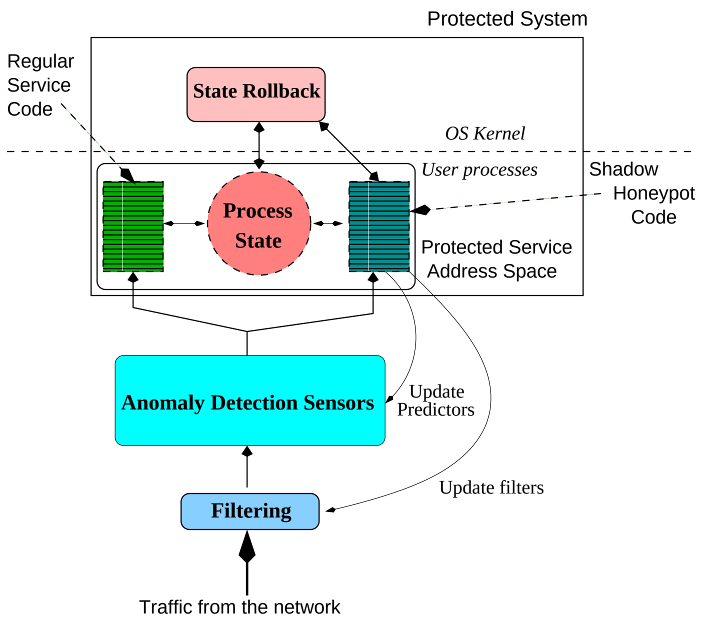

Fig. 1.2. Shadow Honeypot architecture.

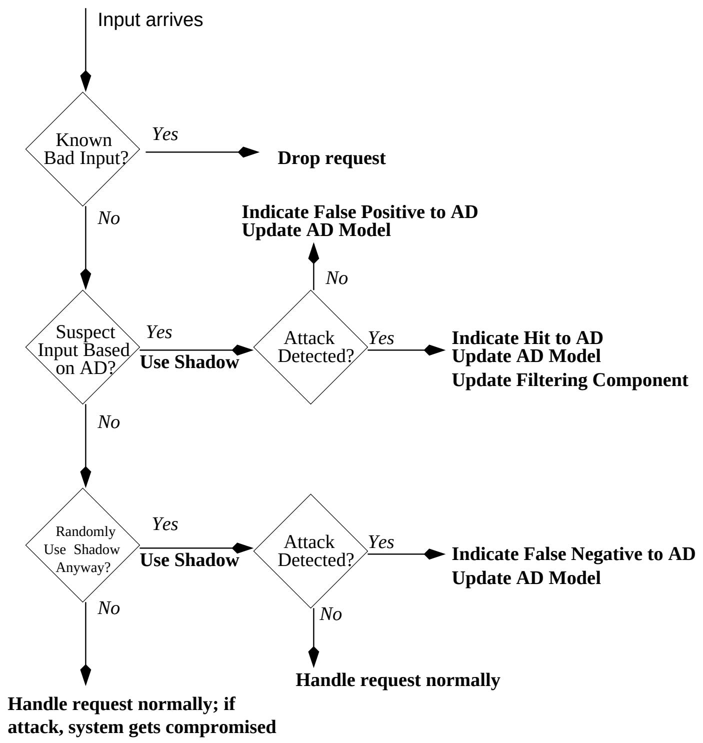

Fig. 1.3. System workflow.

Depending on this prediction by the anomaly detectors, the system invokes either the regular instance of the application or its *shadow*. The shadow is an instrumented instance of the application that can detect specific types of failures and rollback any state changes to a known (or presumed) good state, *e.g.,* before the malicious request was processed. Because the shadow is (or should be) invoked relatively infrequently, we can employ computationally expensive instrumentation to detect attacks. The shadow and the regular application fully share state, to avoid attacks that exploit differences between the two; we assume that an attacker can only interact with the application through the filtering and AD stages, *i.e.,* there are no side-channels. The level of instrumentation used in the shadow depends on the amount of latency we are willing to impose on suspicious traffic (whether truly malicious or misclassified legitimate traffic). In our reference implementation, described in [5], we focus on memory-violation attacks, but any attack that can be determined algorithmically can be detected and recovered from, at the cost of increased complexity and potentially higher latency.

If the shadow detects an actual attack, we notify the filtering component to block further attacks. If no attack is detected, we update the prediction models used by the anomaly detectors. Thus, our system could in fact self-train and fine-tune itself using verifiably bad traffic and known mis-predictions, although this aspect of the approach is outside the scope of the present paper.

As we mentioned above, shadow honeypots can be integrated with servers as well as clients. In this work, we focus our attention on tight coupling with both server and client applications, where the shadow resides in the same address space as the protected application.

- **Tightly coupled with server** This is the most practical scenario, in which we protect a server by diverting suspicious requests to its shadow. The application and the honeypot are tightly coupled, mirroring functionality and state. We have implemented this configuration with the Apache web server, described in [5].
- **Tightly coupled with client** Unlike traditional honeypots, which remain idle while waiting for active attacks, this scenario targets passive attacks, where the attacker lures a victim user to download data containing an attack, as with the recent buffer overflow vulnerability in Internet Explorer's JPEG handling. In this scenario, the context of an attack is an important consideration in replaying the attack in the shadow. It may range from data contained in a single packet to an entire flow, or even set of flows. Alternatively, it may be defined at the application layer. For our testing scenario, specifically on HTTP, the request/response pair is a convenient context.
- **Loosely coupled with server** In this scenario, we detect novel attacks against protected servers by diverting suspicious requests to shadow honeypots. The application is not tightly coupled with the shadow honeypot system, so multiple versions of the application may have to be maintained, and its exact configuration is not known. The requests are captured using passive monitoring and no attempt is made to prevent the attack. This approach has the benefit of being able to "outsource" the detection of vulnerabilities to third entities, potentially taking advantage of economies of scale.
- **Loosely coupled with client** For this approach, suspicious flows are redirected to loosely coupled (do not share state or configuration) versions of an application. One can envision protected client farms where suspicious traffic is tested against multiple versions of the application one is trying to protect.

Tight coupling assumes that the application can be modified. The advantage of this configuration is that attacks that exploit differences in the state of the shadow *vs.* the application itself become impossible. However, it is also possible to deploy shadow honeypots in a *loosely coupled* configuration, where the shadow resides on a different system and does not share state with the protected application. The advantage of this configuration is that management of the shadows can be "outsourced" to a third entity as a service.

Note that the filtering and anomaly detection components can also be tightly coupled with the protected application, or may be centralized at a natural aggregation point in the network topology (*e.g.,* at the firewall).

Finally, it is worth considering how our system would behave against different types of attacks. For most attacks we have seen thus far, once the AD component has identified an anomaly and the shadow has validated it, the filtering component

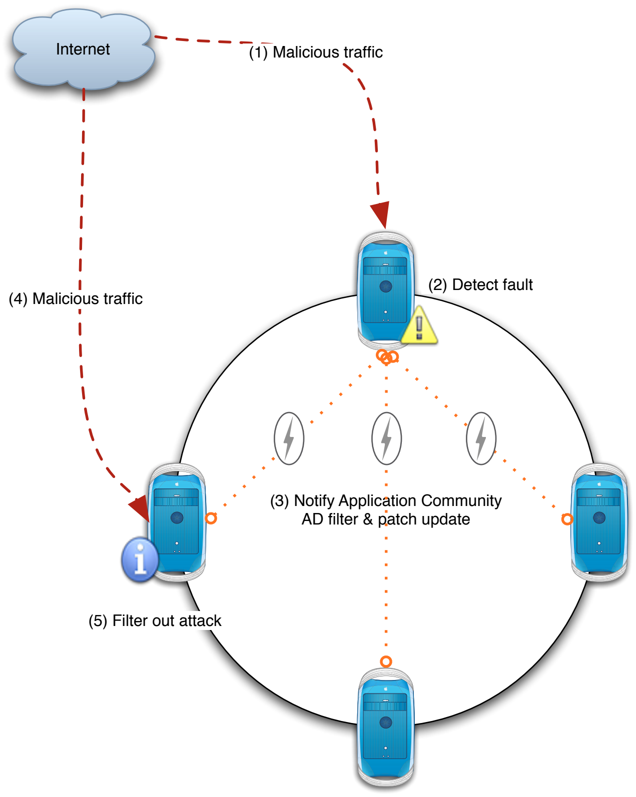

Fig. 1.4. Application Community workflow

will block all future instances of it from getting to the application. However, we cannot depend on the filtering component to prevent polymorphic or metamorphic [51] attacks. For low-volume events, the cost of invoking the shadow for each attack may be acceptable. For high-volume events, such as a Slammer-like outbreak, the system will detect a large number of correct AD predictions (verified by the shadow) in a short period of time; should a configurable threshold be exceeded, the system can enable filtering at the second stage, based on the unverified verdict of the anomaly detectors. Although this will cause some legitimate requests to be dropped, this could be acceptable for the duration of the incident. Once the number of (perceived) attacks seen by the ADS drop beyond a threshold, the system can revert to normal operation.

Application Communities

Using shadow honeypots in a collaborative distributed environment presents another set of interesting tradeoffs and deployment configurations. For this purpose, we explore the use of shadow honeypots with Application Communities [21].

Application Communities (AC) are a collection of almost-identical instances of the same application running autonomously across a wide area network. Members of an AC collaborate in identifying previously unknown (zero day) flaws/attacks and exchange information so that such failures are prevented from re-occurring. Individual members may succumb to new flaws; however, over time the AC should converge to a state of immunity against that specific fault. The system learns new faults and adapts to them, exploiting the AC size to achieve both coverage (in detecting faults) and fairness (in distributing the monitoring task).

Shadow honeypots, in collaboration with Application Communities, provides a systemic framework where the cost of validating false positives is amortized across a large number of hosts and collaboration in anomaly detection can result in more robust vulnerability sensing. Shadow honeypots and AC's can be used in both tightly and loosely coupled configurations. As illustrated in 1.4, each host in an application community participates in protecting portions of an application. For example, if an AC is comprised of four nodes, one (naive) way to split the monitoring cost would be to assign each node responsibility for 25% of the code. For a detailed analysis on work distribution mechanisms and fairness measures, refer to [21]. AC hosts are solely responsible for dealing with their traffic. Traffic that is tagged suspicious by the local AD is processed either by the local instance of the shadow code in a tightly coupled scenario or forwarded to a third party in loosely coupled configuration.

In the tightly coupled scenario, suspicious requests are processed locally by a shadow version of the code that monitors a subset of the application code. This translates into a lower per host cost for processing false positives but relies on collaboration and the size of an AC to achieve both coverage and low overhead. If a vulnerability is detected, it is first processed locally and then information pertinent to the attack is propagated to the AC. In more detail, given a fault, the application is modified so that it can both detect and recover from any future manifestations of the specific fault and the anomaly detectors are updated with signatures derived from the attack vector. This information, along with the attack vector, is then dispatched to the rest of the AC, where hosts can independently validate the vulnerability.

Similarly, for the loosely couple configuration, suspicious requests to a host are sent to a remote sensor that is responsible for monitoring one portion of the application. If the remote sensor detects a fault, the vulnerability specific information is reported back to the host that will, in turn, update the anomaly detector filters and update the application with a version that is protected against the specific fault. At that point, information derived from the remote sensor is communicated to the rest of the AC.

1.3 Limitations

There are three limitations of the shadow honeypot design presented in this paper that we are aware of. First, the effectiveness of the rollback mechanism depends on the detection and recovery capabilities of the vulnerability sensor, and the latency of the detector. The detector used in [5] can instantly detect attempts to overwrite a buffer, and therefore the system cannot be corrupted. Other detectors [41], however, may have higher latency, and the placement of commit calls is critical to recovering from the attack. Depending on the detector latency and how it relates to the cost of implementing rollback, one may have to consider different approaches. The trade-offs involved in designing such mechanisms are thoroughly examined in the fault-tolerance literature (c.f. [15]).

Second, the loosely coupled client shadow honeypot is limited to protecting against relatively static attacks. The honeypot cannot effectively emulate user behavior that may be involved in triggering the attack, for example, through DHTML or Javascript. The loosely coupled version is also weak against attacks that depend on local system state on the user's host that is difficult to replicate. This is not a problem with tightly coupled shadows, because we accurately mirror the state of the real system. In some cases, it may be possible to mirror state on loosely coupled shadows as well, but we have not considered this case in the experiments presented in this paper.

Finally, we have not explored in depth the use of feedback from the shadow honeypot to tune the anomaly detection components. Although this is likely to lead to substantial performance benefits, we need to be careful so that an attacker cannot launch blinding attacks, *e.g.,* "softening" the anomaly detection component through a barrage of false positives before launching a real attack.

1.4 Related Work

Much of the work in automated attack reaction has focused on the problem of network worms, which has taken truly epidemic dimensions (pun intended). For example, the system described in [60] detects worms by monitoring probes to unassigned IP addresses ("dark space") or inactive ports and computing statistics on scan traffic, such as the number of source/destination addresses and the volume of the captured traffic. By measuring the increase on the number of source addresses seen in a unit of time, it is possible to infer the existence of a new worm when as little as 4% of the vulnerable machines have been infected. A similar approach for isolating infected nodes inside an enterprise network [46] is taken in [16], where it was shown that as little as 4 probes may be sufficient in detecting a new port-scanning worm. [58] describes an approximating algorithm for quickly detecting scanning activity that can be efficiently implemented in hardware. [38] describes a combination of reverse sequential hypothesis testing and credit-based connection throttling to quickly detect and quarantine local infected hosts. These systems are effective only against scanning worms (not topological, or "hit-list" worms), and rely on the assumption that most scans will result in non-connections. As such, they as susceptible to false positives, either accidentally (*e.g.,* when a host is joining a peer-to-peer network such as Gnutella, or during a temporary network outage) or on purpose (*e.g.,* a malicious web page with many links to images in random/not-used IP addresses). Furthermore, it may be possible for several instances of a worm to collaborate in providing the illusion of several successful connections, or to use a list of *known repliers* to blind the anomaly detector. Another algorithm for finding fast-spreading worms using 2-level filtering based on sampling from the set of distinct source-destination pairs is described in [54].

[59] correlates DNS queries/replies with outgoing connections from an enterprise network to detect anomalous behavior. The main intuition is that connections due to random-scanning (and, to a degree, hit-list) worms will not be preceded by DNS transactions. This approach can be used to detect other types of malicious behavior, such as mass-mailing worms and network reconnaissance.

[18] describes an algorithm for correlating packet payloads from different traffic flows, towards deriving a worm signature that can then be filtered [24]. The technique is promising, although further improvements are required to allow it to operate in real time. Earlybird [42] presents a more practical algorithm for doing payload sifting, and correlates these with a range of unique sources generating infections and destinations being targeted. However, polymorphic and metamorphic worms [51] remain a challenge; Spinelis [44] shows that it is an NP-hard problem. Buttercup [28] attempts to detect polymorphic buffer overflow attacks by identifying the ranges of the possible return memory addresses for existing buffer overflow vulnerabilities. Unfortunately, this heuristic cannot be employed against some of the more sophisticated overflow attack techniques [30]. Furthermore, the false positive rate is very high, ranging from 0.01% to 1.13%. Vigna *et al.* [55] discuss a method for testing detection signatures against mutations of known vulnerabilities to determine the quality of the detection model and mechanism. Polygraph [27] attempts to detect polymorphic exploits by identifying common invariants among the various attack instances, such as return addresses, protocol framing and poor obfuscation. Toth and Kruegel [52] propose to detect buffer overflow payloads (including previously unseen ones) by treating inputs received over the network as code fragments. The use restricted symbolic execution to show that legitimate requests will appear to contain relatively short sequences of valid *x86* instruction opcodes, compared to attacks that will contain long sequences. They integrate this mechanism into the Apache web server, resulting in a small performance degradation. STRIDE [4] is a similar system that seeks to detect polymorphic NOP-sleds in buffer overflow exploits. [29] describes a hybrid polymorphic-code detection engine that combines several heuristics, including NOP-sled detector and abstract payload execution.

HoneyStat [13] runs sacrificial services inside a virtual machine, and monitors memory, disk, and network events to detect abnormal behavior. For some classes of attacks (*e.g.,* buffer overflows), this can produce highly accurate alerts with relatively few false positives, and can detect zero-day worms. Although the system only protects against scanning worms, "active honeypot" techniques [62] may be used to make it more difficult for an automated attacker to differentiate between HoneyStats and real servers. FLIPS (Feedback Learning IPS) [22] is a similar hybrid approach that incorporates a supervision framework in the presence of suspicious traffic. Instruction-set randomization is used to isolate attack vectors, which are used to train the anomaly detector. Shadow honeypots [5] combine the best features found in anomaly detectors and honeypots to create what an application-aware network intrusion detection system. The anomaly detectors differentiate between trusted and untrusted traffic; trusted traffic is processed normally whilst untrusted traffic is forwarded to a protected instance of the application, its "shadow." The system provides an elegant way to deal with false positives, since all requests are processed albeit some incur additional latency. The authors of [14] propose to enhance NIDS alerts using host-based IDS information. Nemean [63] is an architecture for generating semantics-aware signatures, which are signatures aware of protocol semantics (as opposed to general byte strings). Shield [56] is a mechanism for pushing to workstations vulnerability-specific, application-aware filters expressed as programs in a simple language. These programs roughly mirror the state of the protected service, allowing for more intelligent application of content filters, as opposed to simplistic payload string matching.

The Internet Motion Sensor [6] is a distributed blackhole monitoring system aimed at measuring, characterizing, and tracking Internet-based threats, including worms. [11] explores the various options in locating honeypots and correlating their findings, and their impact on the speed and accuracy in detecting worms and other attacks. [33] shows that a distributed worm monitor can detect non-uniform scanning worms two to four times as fast as a centralized telescope [25], and that knowledge of the vulnerability density of the population can further improve detection time. However, other recent work has shown that it is relatively straightforward for attackers to detect the placement of certain types of sensors [7, 39]. Shadow Honeypots [5] are one approach to avoiding such mapping by pushing honeypot-like functionality at the end hosts.

The Worm Vaccine system [40] proposes the use of honeypots with instrumented versions of software services to be protected, coupled with an automated patch-generation facility. This allows for quick ($<$ 1 minute) fixing of buffer overflow vulnerabilities, even against zero-day worms, but depends on scanning behavior on the part of worms.

The HACQIT architecture [17, 36, 34, 35] uses various sensors to detect new types of attacks against secure servers, access to which is limited to small numbers of users at a time. Any deviation from expected or known behavior results in the possibly subverted server to be taken off-line. A sandboxed instance of the server is used to conduct "clean room" analysis, comparing the outputs from two different implementations of the service (in their prototype, the Microsoft IIS and Apache web servers were used to provide application diversity). Machine-learning techniques are used to generalize attack features from observed instances of the attack. Content-based filtering is then used, either at the firewall or the end host, to block inputs that may have resulted in attacks, and the infected servers are restarted. Due to the feature-generalization approach, trivial variants of the attack will also be caught by the filter. [53] takes a roughly similar approach, although filtering is done based on port numbers, which can affect service availability. Cisco's Network-Based Application Recognition (NBAR) [1] allows routers to block TCP sessions based on the presence of specific strings in the TCP stream. This feature was used to block CodeRed probes, without affecting regular web-server access. Porras *et al.* [31] argue that hybrid defenses using complementary techniques (in their case, connection throttling at the domain gateway and a peer-based coordination mechanism), can be much more effective against a wide variety of worms.

DOMINO [61] is an overlay system for cooperative intrusion detection. The system is organized in two layers, with a small core of trusted nodes and a larger collection of nodes connected to the core. The experimental analysis demonstrates that a coordinated approach has the potential of providing early warning for large-scale attacks while reducing potential false alarms. A similar approach using a DHT-based overlay network to automatically correlate all relevant information is described in [9]. Reference [64] describes an architecture and models for an early warning system, where the participating nodes/routers propagate alarm reports towards a centralized site for analysis. The question of how to respond to alerts is not addressed, and, similar to DOMINO, the use of a centralized collection and analysis facility is weak against worms attacking the early warning infrastructure.

Suh *et al.* [49], propose a hardware-based solution that can be used to thwart control-transfer attacks and restrict executable instructions by monitoring "tainted" input data. In order to identify "tainted" data, they rely on the operating system. If the processor detects the use of this tainted data as a jump address or an executed instruction, it raises an exception that can be handled by the operating system. The authors do not address the issue of recovering program execution and suggest the immediate termination of the offending process. DIRA [43] is a technique for automatic detection, identification and repair of control-hijaking attacks. This solution is implemented as a GCC compiler extension that transforms a program's source code adding heavy instrumentation so that the resulting program can perform these tasks. The use of checkpoints throughout the program ensures that corruption of state can be detected if control sensitive data structures are overwritten. Unfortunately, the performance implications of the system make it unusable as a front line defense mechanism. Song and Newsome [26] propose dynamic taint analysis for automatic detection of overwrite attacks. Tainted data is monitored throughout the program execution and modified buffers with tainted information will result in protection faults. Once an attack has been identified, signatures are generated using automatic semantic analysis. The technique is implemented as an extension to Valgrind and does not require any modifications to the program's source code but suffers from severe performance degradation. One way of minimizing this penalty is to make the CPU aware of memory tainting [10]. Crandall *et al.* report on using a taint-based system for capturing live attacks in [12].

The Safe Execution Environment (SEE) [50] allows users to deploy and test untrusted software without fear of damaging their system. This is done by creating a virtual environment where the software has read access to the real data; all writes are local to this virtual environment. The user can inspect these changes and decide whether to commit them or not. We envision use of this technique for unrolling the effects of filesystem changes in our system, as part of our future work plans. A similar proposal is presented in [23] for executing untrusted Java applets in a safe "playground" that is isolated from the user's environment.

1.5 Conclusion

We have described a novel approach to dealing with zero-day attacks by combining features found today in honeypots and anomaly detection systems. The main advantage of this architecture is providing system designers the ability to fine tune systems with impunity, since any false positives (legitimate traffic) will be filtered by the underlying components.

We have implemented this approach in an architecture called Shadow Honeypots. In this approach, we employ an array of anomaly detectors to monitor and classify all traffic to a protected network; traffic deemed anomalous is processed by a shadow honeypot, a protected instrumented instance of the application we are trying to protect. Attacks against the shadow honeypot are detected and caught before they infect the state of the protected application. This enables the system to implement policies that trade off between performance and risk, retaining the capability to re-evaluate this trade-off effortlessly. We also explore the use of Shadow Honeypots in Application Communities where, the cost of instrumentation is spread across numerous hosts and anomaly detector models can be updated to reflect the findings of hosts that service a variety of different traffic flows.

Finally, the preliminary performance experiments indicate that despite the considerable cost of processing suspicious traffic on our Shadow Honeypots and overhead imposed by instrumentation, our system is capable of sustaining the overall workload of protecting services such as a Web server farm, as well as vulnerable Web browsers. In the future, we expect that the impact on performance can be minimized by reducing the rate of false positives and tuning the AD heuristics using a feedback loop with the shadow honeypot. Our plans for future work also include evaluating different components and extending the performance evaluation.

References

1. Using Network-Based Application Recognition and Access Control Lists for Blocking the "Code Red" Worm at Network Ingress Points. Technical report, Cisco Systems, Inc.
2. CERT Advisory CA-2001-19: 'Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. `http://www.cert.org/advisories/CA-2001-19.html`, July 2001.
3. Cert Advisory CA-2003-04: MS-SQL Server Worm. `http://www.cert.org/advisories/CA-2003-04.html`, January 2003.
4. P. Akritidis, E. P. Markatos, M. Polychronakis, and K. Anagnostakis. STRIDE: Polymorphic Sled Detection through Instruction Sequence Analysis. In *Proceedings of the 20th IFIP International Information Security Conference (IFIP/SEC)*, June 2005.

5. K. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting Targetted Attacks Using Shadow Honeypots. In *Proceedings of the* 14th *USENIX Security Symposium*, pages 129–144, August 2005.
6. 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 *ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 167–179, February 2005.
7. J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet Sensors With Probe Response Attacks. In *Proceedings of the* 14th *USENIX Security Symposium*, pages 193–208, August 2005.
8. M. Bhattacharyya, M. G. Schultz, E. Eskin, S. Hershkop, and S. J. Stolfo. MET: An Experimental System for Malicious Email Tracking. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 1–12, September 2002.
9. M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen. Collaborative Internet Worm Containment. *IEEE Security & Privacy Magazine*, 3(3):25–33, May/June 2005.
10. S. Chen, J. Xu, N. Nakka, Z. Kalbarczyk, and C. Verbowski. Defeating Memory Corruption Attacks via Pointer Taintedness Detection. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 378–387, June 2005.
11. E. Cook, M. Bailey, Z. M. Mao, and D. McPherson. Toward Understanding Distributed Blackhole Placement. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, pages 54–64, October 2004.
12. J. R. Crandall, S. F. Wu, and F. T. Chong. Experiences Using Minos as a Tool for Capturing and Analyzing Novel Worms for Unknown Vulnerabilities. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2005.
13. D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honepots. In *Proceedings of the* 7th *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 39–58, October 2004.
14. H. Dreger, C. Kreibich, V. Paxson, and R. Sommer. Enhancing the Accuracy of Network-based Intrusion Detection with Host-based Context. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2005.
15. E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.
16. J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.
17. J. E. Just, L. A. Clough, M. Danforth, K. N. Levitt, R. Maglich, J. C. Reynolds, and J. Rowe. Learning Unknown Attacks – A Start. In *Proceedings of the* 5th *International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.
18. H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proceedings of the* 13th *USENIX Security Symposium*, pages 271–286, August 2004.
19. C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10th *ACM Conference on Computer and Communications Security (CCS)*, pages 251–261, October 2003.
20. J. G. Levine, J. B. Grizzard, and H. L. Owen. Using Honeynets to Protect Large Enterprise Networks. *IEEE Security & Privacy*, 2(6):73–75, November/December 2004.
21. M. Locasto, S. Sidiroglou, and A. D. Keromytis. Application Communities: Using Monoculture for Dependability. In *Proceedings of the* 1st *Workshop on Hot Topics in System Dependability (HotDep)*, pages 288–292, June 2005.

22. M. Locasto, K. Wang, A. Keromytis, and S. Stolfo. FLIPS: Hybrid Adaptive Intrusion Prevention. In *Proceedings of the* 8th *Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2005.
23. D. Malkhi and M. K. Reiter. Secure Execution of Java Applets Using a Remote Playground. *IEEE Trans. Softw. Eng.*, 26(12):1197–1209, 2000.
24. D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of the IEEE Infocom Conference*, April 2003.
25. D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. In *Proceedings of the* 10th *USENIX Security Symposium*, pages 9–22, August 2001.
26. J. Newsome and D. Dong. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the* 12th *ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 221–237, February 2005.
27. J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *Proceedings of the IEEE Security & Privacy Symposium*, pages 226–241, May 2005.
28. A. Pasupulati, J. Coit, K. Levitt, S. F. Wu, S. H. Li, J. C. Kuo, and K. P. Fan. Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities. In *Proceedings of the Network Operations and Management Symposium (NOMS)*, pages 235–248, vol. 1, April 2004.
29. U. Payer, P. Teufl, and M. Lamberger. Hybrid Engine for Polymorphic Shellcode Detection. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2005.
30. J. Pincus and B. Baker. Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overflows. *IEEE Security & Privacy*, 2(4):20–27, July/August 2004.
31. P. Porras, L. Briesemeister, K. Levitt, J. Rowe, and Y.-C. A. Ting. A Hybrid Quarantine Defense. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, pages 73–82, October 2004.
32. N. Provos. A Virtual Honeypot Framework. In *Proceedings of the* 13th *USENIX Security Symposium*, pages 1–14, August 2004.
33. M. A. Rajab, F. Monrose, and A. Terzis. On the Effectiveness of Distributed Worm Monitoring. In *Proceedings of the* 14th *USENIX Security Symposium*, pages 225–237, August 2005.
34. J. Reynolds, J. Just, E. Lawson, L. Clough, and R. Maglich. On-line Intrusion Protection by Detecting Attacks with Diversity. In *Proceedings of the* 16th *Annual IFIP 11.3 Working Conference on Data and Application Security Conference*, April 2002.
35. J. C. Reynolds, J. Just, L. Clough, and R. Maglich. On-Line Intrusion Detection and Attack Prevention Using Diversity, Generate-and-Test, and Generalization. In *Proceedings of the* 36th *Annual Hawaii International Conference on System Sciences (HICSS)*, January 2003.
36. J. C. Reynolds, J. Just, E. Lawson, L. Clough, and R. Maglich. The Design and Implementation of an Intrusion Tolerant System. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2002.
37. M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA*, November 1999. (software available from *http://www.snort.org/*).
38. S. E. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *Proceedings of the* 7th *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 59–81, October 2004.

39. Y. Shinoda, K. Ikai, and M. Itoh. Vulnerabilities of Passive Internet Threat Monitors. In *Proceedings of the* 14th *USENIX Security Symposium*, pages 209–224, August 2005.
40. S. Sidiroglou and A. D. Keromytis. A Network Worm Vaccine Architecture. In *Proceedings of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, pages 220–225, June 2003.
41. S. Sidiroglou, M. E. Locasto, S. W. Boyd, and A. D. K. omytis. Building A Reactive Immune System for Software Services. In *Proceedings of the* 11th *USENIX Annual Technical Conference*, pages 149–161, April 2005.
42. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the* 6th *Symposium on Operating Systems Design & Implementation (OSDI)*, December 2004.
43. A. Smirnov and T. Chiueh. DIRA: Automatic Detection, Identification, and Repair of Control-Hijacking Attacks. In *Proceedings of the* 12th *ISOC Symposium on Network and Distributed System Security (SNDSS)*, February 2005.
44. D. Spinellis. Reliable identification of bounded-length viruses is NP-complete. *IEEE Transactions on Information Theory*, 49(1):280–284, January 2003.
45. L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2003.
46. S. Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Security*, 2005. (to appear).
47. S. Staniford, D. Moore, V. Paxson, and N. Weaver. The Top Speed of Flash Worms. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*, pages 33–42, October 2004.
48. S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the* 11th *USENIX Security Symposium*, pages 149–167, August 2002.
49. G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. *SIGOPS Operating Systems Review*, 38(5):85–96, 2004.
50. W. Sun, Z. Liang, R. Sekar, and V. N. Venkatakrishnan. One-way Isolation: An Effective Approach for Realizing Safe Execution Environments. In *Proceedings of the* 12th *ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 265–278, February 2005.
51. P. Ször and P. Ferrie. Hunting for Metamorphic. Technical report, Symantec Corporation, June 2003.
52. T. Toth and C. Kruegel. Accurate Buffer Overflow Detection via Abstract Payload Execution. In *Proceedings of the* 5th *Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.
53. T. Toth and C. Kruegel. Connection-history Based Anomaly Detection. In *Proceedings of the IEEE Workshop on Information Assurance and Security*, June 2002.
54. S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders. In *Proceedings of the* 12th *ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 149–166, February 2005.
55. G. Vigna, W. Robertson, and D. Balzarotti. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the* 11th *ACM Conference on Computer and Communications Security (CCS)*, pages 21–30, October 2004.
56. H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *Proceedings of the ACM SIGCOMM Conference*, pages 193–204, August 2004.
57. K. Wang and S. J. Stolfo. Anomalous Payload-based Network Intrusion Detection. In *Proceedings of the* 7th *International Symposium on Recent Advanced in Intrusion Detection (RAID)*, pages 201–222, September 2004.

58. N. Weaver, S. Staniford, and V. Paxson. Very Fast Containment of Scanning Worms. In *Proceedings of the* 13th *USENIX Security Symposium*, pages 29–44, August 2004.
59. D. Whyte, E. Kranakis, and P. van Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network. In *Proceedings of the* 12th *ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 181–195, February 2005.
60. J. Wu, S. Vangala, L. Gao, and K. Kwiat. An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 143–156, February 2004.
61. V. Yegneswaran, P. Barford, and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, February 2004.
62. V. Yegneswaran, P. Barford, and D. Plonka. On the Design and Use of Internet Sinks for Network Abuse Monitoring. In *Proceedings of the* 7th *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 146–165, October 2004.
63. V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha. An Architecture for Generating Semantics-Aware Signatures. In *Proceedings of the* 14th *USENIX Security Symposium*, pages 97–112, August 2005.
64. C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms. In *Proceedings of the* 10th *ACM International Conference on Computer and Communications Security (CCS)*, pages 190–199, October 2003.