SweetBait: Zero-Hour Worm Detection and Containment Using Low- and High-Interaction Honeypots

Georgios Portokalidis and Herbert Bos
Department of Mathematics and Computer Science
Vrije Universiteit Amsterdam, The Netherlands
{porto,herbertb}@few.vu.nl

***Abstract*— As next-generation computer worms may spread within minutes to millions of hosts, protection via human intervention is no longer an option. We discuss the implementation of *SweetBait*, an automated protection system that employs low- and high-interaction honeypots to recognise and capture suspicious traffic. After discarding white-listed patterns, it automatically generates worm signatures. To provide a low response time, the signatures may be immediately distributed to network intrusion detection and prevention systems. At the same time the signatures are continuously refined for increased accuracy and lower false identification rates. By monitoring signature activity and predicting ascending or descending trends in worm virulence, we are able to sort signatures in order of urgency. As a result, the set of signatures to be monitored or filtered is managed in such a way that new and very active worms are always included in the set, while the size of the set is bounded. *SweetBait* is deployed on medium sized academic networks across the world and is able to react to zero-day worms within minutes. Furthermore, we demonstrate how globally sharing signatures can help immunise parts of the Internet.**

I. Introduction

As new breeds of worms are expected to spread to millions of hosts in minutes, if not seconds, it is imperative to automate both outbreak detection and response [1], [2]. Worse, in order to be effective the automated system should take appropriate counter measures in a fraction of the time that it takes the worm to spread. Previous attempts to develop such detection systems have built on flow-based anomaly detection, honeypots, and end-host detection [3], [4], [5]. Several projects have addressed the problem of automatic signature detection [6], [7]. Unfortunately, most existing approaches exhibit one or more of the following problems:

1) *False positives*. For any automated response system holds that misclassifying and blocking *bona fide* traffic may result in unleashing a denial of service attack by the defence mechanism.
2) *Instances rather than variations.* Most existing systems extract the signature of an individual worm with no attempt to check whether this is a variation of a worm that was previously detected.
3) *Presence rather than virulence.* Anyone brave enough to connect an unprotected machine to the Internet will soon discover that there are many different worms out there. In addition to an exhaustive list of what worms have been encountered in various places, a security system would benefit from information about the worm activity level. Virulent worms may require more drastic and immediate measures than worms that spread slowly.
4) *Known worms rather than zero-day attacks.*

In *SweetBait* we address the problem of fast worms by means of honeypots. The system also detects worms that spread slowly and even other forms of malware, but these are not its focus. We discuss the design and implementation of an automated response system that aims to protect small and medium sized networks from random IP scanning worms. The size of the networks in focus was motivated by a desire to avoid performance issues that arise with systems on backbone links. Our goal is to automate the procedures of both worm signature generation, and signature distribution. Signatures are distributed both to external network intrusion detection and network intrusion prevention (NID and NIP) systems, and to external host-based intrusion prevention (HIP) systems. At the same time we aim to achieve a low reaction time to new outbreaks. The challenging task of identifying new worms is performed by honeypots. In *SweetBait* the honeypots along with the NID and NIP systems will be managed by a *control centre* (CC), which will be able to respond to outbreaks even when untended. Some contributions of this paper are summarised below:

1) reduce false positives by requiring confirmation from multiple sensors and by 'whitelisting' benevolent traffic;
2) continuously refine worm signatures to provide automated signature revision;
3) employ HID, NID and NIP systems for detection and containment;
4) predict worm aggressiveness by monitoring a worm's activity level;
5) through an open design, allow different types of honeypots to be plugged in;
6) distribute signatures through a *global control centre* (GCC) to all instances of the system to achieve possible immunisation of parts of the Internet.

To prove point (5) we implemented two different signature generators. One is known as *SweetSpot* and is based on a *low-interaction* honeypot similar to honeyd. The other, known

as *Argos*, is still experimental and only partially integrated with *SweetBait*. It consists of an innovative *high-interaction* honeypot on top of an x86 emulator. We chose honeypots rather than network taps for several reasons. First, network administrators feel more comfortable with handing out chunks of unused IP address space than with systems that snoop on user traffic. Second, while it is true that honeypots by themselves never see hit-list worms, this was easily remedied by directing suspicious traffic to the honeypot. Indeed, *Argos* is used to deploy an *advertised honeypot*: unlike most honeypots, we do not try to make *Argos* invisible. On the contrary, it is intended that *Argos* honeypots are linked to actively. For instance, hidden links can be added to webpages that point to *Argos* sensors. While no human would normally follow these links, one should expect a fair amount of benevolent traffic in the from of web crawlers etc. For this reason, the issue of false positives is even more important than for *SweetSpot*. To ensure that we do not generate signatures in response to benevolent traffic, we demand the number of false positives in the intrusion detection component of *Argos* to be zero. At the same time, we demand that the number of false negatives for the types of attack that are recognised by *Argos* to be zero. While we have implemented a functional prototype of *Argos*, we have only recently started to embed it in *SweetBait*. Unlike *SweetSpot*, we have therefore not deployed it beyond our own laboratory testbed.

Even with *SweetSpot* sensors it is possible to capture certain hitlist worms by actively directing traffic to the honeypot. In this case, the NID would function as a two-tier system that uses anomaly detection in the network as a first, and the *SweetSpot* honeypots as a second tier in the detection process. Whenever unusual behaviour is detected, the corresponding traffic is forwarded to honeypots for further analysis. In this way, we preserve the property that all traffic arriving at the honeypot is suspect. This is in contrast to both our *Argos* signature generator and all approaches that protect against attacks at the user's machine, e.g., [8] and our own [9]). These latter approaches must therefore work harder to weed out the false positives.

Note that a third argument is sometimes made in favour of honeypots, namely that random IP scanning worms have been much more popular than hit-list worms and scanning worms are responsible for the fastest spreading behaviour to date. While true, we do not consider this a valid assumption for a future-proof system. For instance, Staniford, Paxson and Weaver have demonstrated that in theory hit-list worms have the potential spread faster than scanning worms [1]. We therefore expect to see more of these attacks in the future.

Currently, we have a fully functional implementation of *SweetBait*/*SweetSpot*, and an experimental prototype of *Argos*, which is in the process of being incorporated into *SweetBait*. In this paper we describe the *SweetBait* architecture, as well as the main components that were implemented. In Section II we will give a detailed description of the system's architecture. In Section III, we outline our implementation. We evaluate the system in Section IV. Since the embedding of *Argos* in *SweetBait* is work in progress, while *SweetBait*/*SweetSpot* has already been actively deployed at several sites around the world, evaluation for both honeypots will be along different lines. Related work is discussed throughout the text and also in Section V. Finally, we will present our conclusions and future work in Section VI.

II. System Overview

SweetBait is comprised of multiple components with distinct roles, which can be roughly classified into two categories: sensors and control elements. Honeypots, intrusion detection and prevention systems are all sensors, while *control centres* and a *global control centre* constitute the control components. The honeypots are set up to receive data destined to nonexistent IP addresses of the corresponding subnet as well as any traffic that is explicitly directed to them as explained in the previous section.

These data are first filtered to exclude any known benevolent traffic patterns. The remainder is treated as of malicious origin and is processed to generate NID signatures that we claim to belong to malware. The generated signatures are then posted to the CC, where they are compared with the ones already known. Based on the incidence reports from multiple locations, the CC decides which signatures to transmit to the NID and NIP components. NID and NIP sensors return feedback to the CC concerning the number of hits for the signatures they have been monitoring or filtering. Finally, the CC is responsible for exchanging signatures and activity statistics with a GCC. The presence of a GCC enables cooperation of instances of *SweetBait* globally, which is necessary to achieve worm containment [1].

A typical configuration of *SweetBait* components is shown in Figure 1. In the remainder of this section we will describe each component in detail.

A. Honeypot sensors

Honeypots are powerful devices for capturing random IP scanning worms. These worms discover new victims to attack by randomly generating IP addresses. We distinguish between two types of honeypots: hidden and advertised.

The IP address of a hidden honeypot is by its nature unadvertised on the Internet, and as such it does not exchange any legitimate data with the rest of the network. We may therefore assume that all traffic destined to it is suspect. By populating dark IP space of a network with hidden honeypots, there is a high probability of finding a scanning worm in its early stages.

For advertised honeypots the assumption that all traffic is suspect does not hold. Hence, it must have a way to separate the good from the bad. As shown in Section III-B, the way we do this in *SweetBait* is by only generating signatures for data that is demonstrably malignant. In essence, we use memory tainting to trigger an alarm if we catch in the act either a stack-smashing/heap overflow or format string attack.

Deploying a honeypot presents us with two further possibilities. The first is to sacrifice a real host running *real*

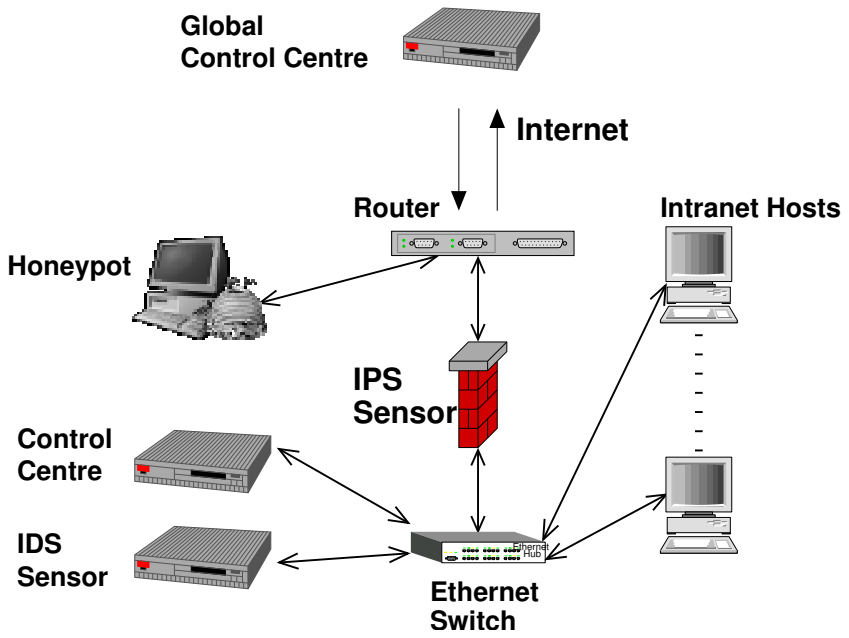

Fig. 1. Architecture overview

services (high-interaction), while the second is to *simulate services* and/or hosts (low-interaction). The former offers high-interactivity with attackers and makes the honeypot almost indistinguishable from other hosts, but it requires additional protection mechanisms such as sandboxing [10], [11]. Since no real services are run, the latter offers a lower level of interaction, but requires less maintenance and supplies a greater degree of security [12], [13]. Also, it was shown in NoSEBrEaK that some high-interaction honeypots can be easily discovered as such by intruders [14].

SweetBait uses both low-interaction and high-interaction honeypots. The main advantages of low-interaction honeypots for *SweetBait* are the low maintenance requirements and security considerations, which makes it easier to deploy. Other projects such as Leurre.com also opted for low-interaction for this reason. The low-interaction honeypot, known as *SweetSpot*, is able to simulate multiple hosts, permitting us to populate unused IP address space easily and to maximise the amount of captured traffic.

The high-interaction honeypot in *SweetBait*, known as *Argos*, is much more complex. It captures both known worms and zero-day attacks. In addition, it can be used to generate both network signatures and signatures for end systems. We will discuss *Argos* in detail in Section III-B.

Even though we consider all traffic received by the *SweetSpot* honeypot suspect, in practice a small amount of non-worm related, or even legitimate traffic is also captured. A well-known source of spurious traffic, for instance, is back-scatter from DDoS attacks. Broadcast messages, or attempts to scan the network are examples of traffic that for our purposes may be ignored. To tackle this we introduce the notion of a *whitelist*: a list consisting of patterns that are considered benevolent, or inapplicable for generating NID signatures. A filter placed at the honeypot rejects all traffic matching a *whitelisted* pattern. The filter can be largely auto-generated by training the system on a network when it is unconnected to the larger Internet. This simple step reduces the number of false postives and increases significantly the accuracy of the signatures that are generated.

Consequently, the sensors process the remainder of incoming data. The way the data is processed is specific to the sensor. The sigantures that are generated, however, must conform to the *SweetBait* format.

For instance, low-interaction *SweetSpot*s scan traffic for repeated byte sequences in an attempt to identify worm propagation and to generate a signature. For the scanning process to be effective it is necessary to utilise stream reconstruction for sequenced, connection-based protocols such as TCP, since the underlying IP layer may deliver packets out of order impeding our capability to identify patterns spanning multiple packets. High-interaction *Argos* honeypots push the data up to the application and triggers a signature generation phase whenever a violation is detected. It is able to generate different types of signature, one of which is a network signature similar to the one generated by *SweetSpot*.

All signatures conform to *SweetBait*'s format. The format consists of a small header identifying the type of signature, the system that generated it, etc. As an example, the following two signatures show different *types* of signatures generated by the two honeypots:

```
SIGNATURE 1
generated by: SweetSpot
source: 10.0.0.1.
destination: 192.168.2.1
type: network signature
subtype: string
description: longest common
             substring
```

```
SIGNATURE 2:
generated by: Argos
source: 10.0.0.2
destination: 192.168.2.2
type: host signature
subtype: string
description: pattern for encry-
             pted IIS data
```

Finally, we transmit the generated signatures from the sensor to the *control centre*, where subsequent actions are taken. The transmission is done in a secure manner that guarantees the authentication of both parts, as well as the integrity of the transmitted data.

B. Network Intrusion Detection and Prevention Sensors

We use network intrusion detection sensors to passively monitor ingress and egress traffic for worms, based on the signatures of type network signature generated by the honeypots. Whenever a signature is matched, the NID sensor reports it by issuing an alert that also includes the IP addresses involved in the transaction. Besides allowing us to quantify a worm's activity, this information also enables us to:

- populate an Internet map with infected IP addresses;
- block infected remote hosts from accessing our network;
- identify infected hosts in our network, and initiate immunisation procedures.

NIP sensors, besides monitoring and reporting worms by issuing alerts, assume the active role of filtering ingress and egress traffic based on signatures of type `'network'`. If initiated before any host in the network has been compromised, blocking worms from entering the network will lead to immunisation. On the other hand, obstructing worms from leaving the network is tantamount to 'team play' on our part that helps contain the worm, and earns time for other networks to raise their defences.

Most NID and NIP systems today are manually updated each time a new worm appears, while alert reports are being used purely for historical purposes. The *SweetBait* sensors are in constant communication with the *control centre*, which is responsible for automatically updating the sensors with the signatures that need to be monitored or filtered, respectively. Additionally, we immediately post the alerts generated by the sensors to the CC for storage, as well as for estimating worm aggressiveness. Again, the communication channel is secure to ensure only authorised access to the CC.

C. Host-based Intrusion Detection and Prevention Sensors

Although *SweetBait* as it is deployed only uses NID and NIP sensors, the version under development employs other sensors as well. For instance, as the *Argos* honeypot generates network signatures as well as host signatures, we are able to send signatures to hosts automatically. While one could, in principle, employ signatures similar to self-certifying alerts as employed in Vigilante, we have focused primarily on host-based *filtering* and left the distribution and verification of signatures for future work.

As *Argos* automatically generates signatures when buffer overflows or format string attacks occur, we have no problem in catching zero-day worms. Generating the signature is more difficult as we shall see in Section III-B.

The sensor that we implemented thus far is really intermediate between a host-based and a network-based, as it is implemented in software on a programmable NIC. It scans all incoming traffic for the occurrence of signature of worms and viruses. It is described in detail in [9].

D. Control Centre

If honeypots and intrusion detection/prevention systems are the arms and legs of *SweetBait* the *control centre* is the brain. The CC collects information from the sensors, processes it and instructs the sensors on future actions. Information exchange between the CC and connected sensors is performed based on a well-defined protocol, decoupling the exchanged data from specific sensor types. The CC gathers two types of incidence reports: network intrusion signatures and alerts.

Signatures are compared with every known signature to detect overlaps. When significantly sized overlaps exist, the received signature is considered to be a *specialisation* of the stored one, and it is stored as a new version. Besides the actual network intrusion signature, the stored data consist of accompanying meta-information. Such data include the timestamp to indicate the time of generation, the source of the signature (e.g., the IP of the honeypot), and various flags to indicate whether it has been verified by an expert to be a valid or a false signature. The latter is essential information that permits humans to affect the decision making process that is described later to increase the effectiveness of the system.

The CC also collects alerts generated by NID and NIP components when network traffic is found to match one of the known signatures. Each alert contains information that identifies the signature, as well as the source and destination IP addresses involved in the data exchange. The information is stored in a database both for auditing and the reasons mentioned in Section II-B. Furthermore, the number of occurrences of a worm in the network is an indication of its aggressiveness, and will be used to classify the signatures based on the threat they pose. We discuss later how administrators may require signatures to be confirmed by multiple sites to reduce the number of false positives.

Besides *gathering* these incidence reports, the CC also *pushes* information to HID, NID and NIP sensors automatically. A sensor that identifies itself as able to perform either intrusion detection or prevention receives periodic reports about what packets it should monitor or block. This automates the deployment of NID signatures, and represents a significant step to zero-hour detection and containment.

In most implementations, the number and size of signatures that each NID or NIP is looking for determines its maximum throughput, therefore we enforce a limit on the size in bytes of the signatures that are pushed to the sensors. We adopt the notion of a *signature budget*, signatures are sorted based on their virulence and we push as many as the budget permits. Additionally, a policy dictates which of these signature are to be filtered. When a signature's activity exceeds a configurable threshold, the sensors will be instructed to filter the offending traffic. Whether a signature is verified to be valid or false, can also be used to exclude false or even unverified signatures from being filtered, preventing in this way the system from accidentally stopping legitimate traffic.

Finally, the CC periodically exchanges information with the *global control centre* (GCC). This includes newly generated signatures, as well as activity statistics of known signatures. The statistics received by the GCC are accumulated with the ones generated locally to determine a worm's aggressiveness. This accumulation ensures that the CC is able to react on a planetary outbreak, even if it has not yet been attacked itself, achieving immunisation of the protected network.

E. Global Control Centre

The detailed specification of a planetary scale centre for worm control is beyond the scope of our work, nevertheless we briefly mention the aspects of such a centre that are necessary for *SweetBait* and that we have implemented. The GCC collects signatures and statistics in a similar way to a CC, with the main difference being the lack of a signature budget when pushing signatures to CCs. Additionally, because of performance, as well as privacy concerns only the number of alerts is exchanged discarding the source and destination IPs information. As a single GCC obviously is susceptible to attacks and could become itself a liability, a distributed solution is preferable both for performance and security reasons.

F. Reducing System Vulnerability

SweetBait has been designed to protect sensors, control elements and the communication channels between them. However, like in most automated response systems, it may be possible to manipulate the signatures that are generated, e.g., by bombarding *SweetSpot* with identical harmless packets. The packets cause the system to incur a high incidence rate for the corresponding signatures, and hence may lead to an automated response that blocks the traffic in the network. In this way attackers might even be able to cause *SweetBait* to block user traffic. To do so, attackers need to discover the honeypot first. Alternatively they could flood the entire network with the traffic, but this would be soon detected. While *whitelisting* helps a little to counter such manipulations, it is by no means sufficient.

The presence of a GCC may be more effective in mitigating these effects by ensuring that attackers cannot easily manipulate activity rates. A worm's activity is determined by accumulating the rates reported by all *SweetBait* instances through the GCC, therefore the effects of such an attack would be dulled.

For stronger protection, *SweetBait* has a configurable parameter that indicates how strongly the incidence reports should be confirmed by other honeypots. As it is much harder to discover $n > 1$ honeypots rather than just a single one, the *SweetBait* CCs may require incidences to be confirmed by at least n sensors, trading response time for robustness.

III. System Implementation

This section discusses the implementation of sensors and CC. We have employed off-the-shelf solutions wherever possible in an attempt to allow already deployed systems to be integrated in *SweetBait*. As we only started to incorporate *Argos* in *SweetBait* recently, and its integration is only partially complete, it was not part of the deployed architecture. In addition, *Argos* employs novel ways of detecting worms and generating signatures and for this reason it will be evaluated along different dimensions than *SweetSpot*.

A. Low-Interaction Honeypot Sensor: SweetSpot

For *SweetSpot* we use *honeyd*, a virtual honeypot framework that provides multiple low-interaction virtual honeypots on a single host [4]. It captures traffic destined to unused IP addresses on the deployed network and supports third party plug-ins that can access and process the captured traffic. Every *SweetSpot* is attached to an operating system (OS) profile that results in the simulation of its TCP stack on established connections. This approach protects the host from tools like xprobe and nmap [15], [16] that fingerprint TCP packets to identify its implementation and expose the host's OS. Besides simulating operating systems, honeyd supports scripts that emulate services such as a web server or a telnet daemon.

For automatically generating NID signatures from the captured traffic, we employ *honeycomb*, a *honeyd* plug-in that scans incoming traffic and detects repeating patterns using the longest common substring (LCS) algorithm [7]. In addition, *honeycomb* performs flow reconstruction, and is able to detect patterns even when they are segmented in multiple IP packets. Signatures are periodically written out to a log file in pseudo-snort rule format along with a timestamp that can be later read and distributed to the CC.

To utilise a filter for *whitelisted* patterns, we developed a new *honeyd* plug-in named *honeybounce*. This plug-in supports a list of rules that specify in snort rule format the patterns to be excluded from the NID signature generation. This is achieved by loading *honeybounce* prior to *honeycomb* and rejecting the matching packets. Since *honeyd* plug-ins do not support packet rejection, we have developed a patch that installs this functionality. Currently, *honeybounce* does not perform flow reconstruction, because of constraints of the *honeyd* plug-in architecture and also to conserve CPU time for pattern detection by *honeycomb*. We do not expect this to become an inconvenience, because of the nature of *whitelisted* traffic, which is benevolent by definition and consists mainly of broadcast and multicast messages originating from the subnet. These messages are mostly small enough to be contained in a single packet, and in all other cases we observed in practice that fragmentation is predictable, being the result of regular fragmentation of a stream into IP packets.

Honeybounce supports filtering of TCP and UDP packets based on exact byte sequences and perl regular expressions. To accelerate the filtering procedure the filters are classified in three categories based on protocol: TCP, UDP or ANY. For each of these classes the filters are hashed based on their destination port number(s), which can also be ANY. Filters are applied sequentially, when a match occurs the procedure is terminated by signalling *honeyd* to reject the packet averting its processing by *honeycomb* and subsequent plug-ins.

The signatures generated by *honeycomb* are read by a signature distribution process that transmits them immediately to the CC. The signatures are first examined to ensure that the content is valid and that they are not older than the last signature received from the CC. This is accomplished by requesting the timestamp of the last signature received from the CC, when first establishing the connection. Such an approach is necessary, because *honeycomb* generates signatures even for obscure protocol flags combinations, and additionally dumps all generated signatures periodically. Finally, SSL is used between the honeypot and the CC to perform authentication between the two using public and private keys, and to ensure that the exchanged data are secure from eavesdropping.

B. High-Interaction Honeypot Sensor: Argos

Argos is much more complex than *SweetSpot* and while a functional prototype exists, we have just started exploring the domain of signature generation. As a result, our current signatures are rather crude, and we expect more results in this area in future work. A detailed discussion of *Argos*' implementation is beyond the scope of this article and is described in a conference paper submitted elsewhere. Note that unlike the signature generator, the intrusion detection part of *Argos* is not a proof of concept solution, but a deployable system with reasonable overhead that runs on commodity hardware and on top of various OSs.

Argos is a high-interaction honeypot based on a modified x86 processor emulator known as Qemu [17]. On top of the emulator we run the OS of our choice. No changes to the OS are required and we can therefore support any OS running on the IA32 architecture. In practice, we have succesfully tested *Argos* with Linux, Windows 2000 and Windows XP. On top of the OS, we run the applications we want to track (e.g., Apache, IIS, etc.). Unlike many other approaches, *Argos* has no knowledge of the applications and protects all code running on top of it, including the kernel.

Misbehaving code is detected at the level of the emulator. We use dynamic taint analysis [18] to detect when a vulnerability is exploited to alter the target's control flow. Dynamic taint analysis aims to identify the illegal uses of unsafe data such as data received from the network. For instance, using values originating in the network as jump targets, function call, or return addresses is considered to be illegal. Additionally, executing data originating from the network is also not allowed to capture attacks that while control flow is not altered, inject arbitrary instructions into locations known to be jump targets.

The signature that is generated by *Argos* is crude but effective. Whenever a violation is detected, we dump to file all tainted pages that correspond to the currently active code (the page tables help us determine wich pages belong to this code). In addition, when we detect illegal use of tainted data, we also dump the jump target (4 bytes on our architecture), and the page containing the jump target.

Furthermore, by inserting our own shellcode in the code that is currently under attack, we unearth relevant information about the process. In the current implementation, we read the process identifier and the executable name, but in the future we plan to extend this to open files, open sockets, etc. In other words, we inject our own 'attack' to gather useful information about the real attack. All this data is written to file for analysis (e.g., by a human expert) and automatic signature generation.

Our signature generator scans the network traces towards this application for the occurrence of the jump target. In case of TCP traffic, the network stream is first reconstructed. Next, when the jump target is found, we scan for information around the jump target. Thus we create a maximum-sized network signature for the attack that is subsequently sent to the CC.

If we did not find the jump target, it probably means that the traffic is encrypted. Current implementation work aims to cope with this type of attack by interposing between the application and well-known encryption/decryption libraries. For *Argos*, we are implementing a handler that replays the trace to the application after restoring the application in a clean state. By placing a wrapper function between the application and the encryption library to scan all data flowing towards the application, we can apply the same steps as for unencrypted traffic, except that we generate a signature of a different type. At this point we have implemented the interposers and checked that it is possible to find a signature in this way.

We stress that *Argos* is only a first stab at an automated defense against some really complex attacks and by no means a mature IDS. For instance, in the current implementation we do not yet provide secure communication to the CC, we do not automatically roll back services on the *Argos* hosts after an attack was detected, etc. All these are fairly prosaic problems which prevent the system as is from being widely deployed. However, we believe that *Argos* represents an important step as it explores an extreme in the design space of intrusion detection systems: detecting zero-day attacks and may in the future even handle encrypted attacks. Furthermore, we will show that it is able to detect reliably all buffer overflow attacks. It flags no false positives and so far we have not found any false negatives either.

C. Host Intrusion Detection Sensor

The HID sensor we developed so far includes a pattern matching engine implemented in software on a programmable network card. This solution, known as *cardguard*, is described in detail in [9] and consist of a parallel implementation of the well-known Aho-Corasick pattern matching algorithm on Intel IXP1200 network cards. All traffic that arrives at the host is subjected to a payload scan for signatures derived from the snort rule set [19]. For TCP flows, Cardguard first employs stream reconstruction.

Note that all checks are performed before the traffic even reaches the host CPU. Besides off-loading the host CPU this has an additional advantage. Administrators may be reluctant to trust the software running on a host CPU. By placing the HIP sensor in the NIC, shielded from the end-user may be considered more acceptable from a political viewpoint.

The rules in *cardguard* are encoded as a deterministic finite automaton (DFA) that is generated off-line. A packet or TCP

stream is matched one byte at a time and for each byte the DFA incurs a single state transition. Because all rules are encoded in a single DFA, one state transition matches all rules at once. The overhead incurred is therefore proportional to the size of the packets rather than the number or size of the rules. As there are many thousands of rules already, this is a very desirable property.

Cardguard uses the measured locality of reference in DFA accesses to steer the memory layout of the DFA. In other words, states that are needed frequently are placed in highspeed memory (on-chip instruction store), while less frequently states are placed in off-chip SRAM or DRAM. While IXP1200s are now considered obsolete, the performance of *cardguard* (600 Mbps for UDP, 100 Mbps for TCP) is quite acceptable for most end-host systems.

D. Network Intrusion Detection and Prevention Sensor

Snort [19] is one of the most popular open source NID systems, and is deployed in many networks. This along with the fact that *honeycomb* generates signatures in snort rule format motivated the adoption of snort as the base of NID sensors. Snort scans received traffic for a set of rules and generates an alert each time a match occurs. These alerts are logged in a file and subsequently transmitted to the CC. The information contained in an alert includes a custom annotation, which we use to identify the rule that caused it, and the involved IP addresses. Such an alert is shown in Figure 2. Snort can also react (in a passive way) when a TCP flow has been found to match a rule by using control packets to terminate it. This is accomplished by transmitting a TCP FIN packet to both ends of a TCP flow, when a corresponding rule is matched. Unfortunately, in the case of worms such a mechanism is not sufficient, since the original packets containing the worm have already reached their destination, and have probably infected it.

Since snort is not deployed in-line and does not offer an efficient protection mechanism, along with the lack of open NIP alternatives, we implemented a simple NIP system based on Linux netfilter (`http://www.netfilter.org`). Netfilter on a Linux router permits us to intercept packets before being routed, and thus filter them at the point of entry in the network.

We developed a Linux kernel module, named `CBFilter`, based on Netfilter to perform content-based filtering. The module scans for byte sequences in packets' payload using the well-known Aho-Corasick algorithm [20]. If (a) a packet arrives with a protocol and portnumber combination that is specified 'to be checked', and (b) the payload matches a target pattern corresponding to one of the rules, and (c) the protocol and destination port number also match, then the packet is dropped. We have chosen to scan first the payload of a packet and then check the protocol and port number to avoid instantiating multiple search trees for each protocol and port number pair. Such an approach would diminish the benefits of using the Aho-Corasick algorithm, reducing performance to that of a serial search algorithm. Note that even though the algorithm is the most effective we could use, this procedure remains computationally expensive.

`CBFilter` is controlled from user-space over a device file. A process can instruct the module to load new filters, start and stop filtering, as well as recover statistics. Statistics include the number of packets filtered using a specific rule, but not the source and destination addresses involved due to performance restrictions. Each time statistics are retrieved, the corresponding counter is reset. prototype

The alerts generated by snort and `CBFilter` are collected by a signature distribution process that transmits them to the CC. The same process also listens for signature updates from the CC, and applies them to snort and `CBFilter`. To minimise data transmission, the process supports alerts caching: aggregation of the alerts generated by each rule, and periodic transmission of the number of hits incurred during each period. This is useful when the number of alerts is sufficiently large to approach saturation either of the connection with the CC, or of the CC itself. The aggregation occurs at the expense of detailed information that may be used for auditing, as the IP addresses are not included with the aggregate alerts. As in all cases, we use SSL on this connection for authentication and security purposes.

E. Control Centre

We implemented CC as a multi-threaded server that handles multiple concurrently connected sensors, and uses a PostgreSQL database for storing its data (`http://www.postgresql.org`). The CC collects two types of information: signatures and alerts. When a new signature N is received it is first compared with every stored signature S, sharing the same protocol and destination port number. The comparison is done in a signature-type-specific manner. Currently, we have only used the LCS algorithm for all signatures of type string. If an overlap O exists, then the following applies:

1) **Specialisation:** the length of O is at least $X\%$ of the length of N.
   O will be treated as a new version of the stored signature S. In practice we found an initial value of $X = 85$ to perform well, as it leaves space for the generation of more specialised signatures, while protecting the system from misclassifying a new worm signature N as a new version of a stored signature S.
2) **Generalisation:** the length of O is at least $Y\%$ of the length of S.
   This rule was introduced to keep the system consistent when a new honeypot sensor is introduced or an already running one is restarted. The honeypot sensors do not hold persistent information regarding previously generated signatures; when restarted in an attempt to generate signatures as soon as possible they will generate signatures more generic than the ones already stored at the CC. These signatures will be large enough to evade the first rule, but will be captured by this one. The value of Y should be just below 100, or even 100 to completely eliminate the possibility of missing valid

```
[**] [1:2003:4] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
08/24-16:03:13.805589 XXX.XXX.X.XX:1178 -> XXX.XXX.XX.XX:1434
UDP TTL:108 TOS:0x0 ID:30134 IpLen:20 DgmLen:404
Len: 376
[Xref => http://vil.nai.com/vil/content/v_99992.htm]
[Xref => http://www.securityfocus.com/bid/5311]
[Xref => http://www.securityfocus.com/bid/5310]
```

Fig. 2. Example of a Snort alert

new signatures. In practice, we found a value of 95 to be sensible and effective.

In both cases, if O is identical with S, it is discarded. Furthermore, to avoid over-specialisation of signatures and unreasonable signature lengths, we also introduce a minimum and maximum. If the length of N or O do not fall within these limits it is also discarded. Process the alerts is straightforward: whenever an alert is received the activity counter of the corresponding signature is increased, and the involved IPs are stored in the database.

The CC periodically updates the NID, NIP and HID sensors with the set of signatures that should be monitored and filtered respectively. Because we enforce a budget on the maximum size of the signature set deployed on these sensors, we need to sort them based on their expected aggressiveness. To quantify this, we selected the exponentially weighted moving average of the number of alerts generated by each signature on each period. It is defined as

$$m' = w \times a + (1 - w) \times m \tag{1}$$

where: m' is the new value, m is the previous value, a is the number of alerts this period, and w is the weight (a configurable parameter). Selecting a value for the weight $0 < w \leq 1$ configures m to follow more or less aggressively the recent changes in activity levels. In practice, we found that values less then 0.5 are not very useful. The value m is used to predict future values of a worm's aggressiveness. The value of a signature's activity A that is eventually used by *SweetBait* is biased towards specific destination ports and protocols:

$$A = m \times portbias \times protocolbias \tag{2}$$

This approach allows us, for instance, to react more aggressively to UDP than TCP worms, and with caution to signatures involving web or mail services. This is also useful as some ports (e.g., ports 139 and 80) and protocols are much more frequently attacked (or scanned) than others [21].

The signatures are subsequently transmitted to the NID, NIP and HID sensors, starting with new signatures, and proceeding with signatures that have the largest activity value, going as far as the signature budget allows. Signatures with a value of A larger than the filtering threshold are transmitted to the sensors with the indication that they should be filtered, unless the administrator of the system has requested that only verified ones should be filtered.

Finally, the CC periodically contacts the GCC to exchange signatures and global activity statistics. The received statistics are aggregated with the local ones to provide new values of a and consequently A. Again, SSL is used for communication between CC and GCC.

F. Global Control Centre

The *global control centre* is a stripped-down version of the CC described above. It is a multi-threaded server that handles multiple connections from CCs, and exchanges signatures and statistics. Signature specialisation is done as described in Section III-E. Activity statistics received by the CCs are aggregated, and are periodically cleared to avoid stale values from inhibiting the ability of detecting new outbreaks.

IV. Experimental Evaluation

To evaluate *SweetBait* we deployed it at four different sites: Vrije Universiteit in Amsterdam, ICS FORTH in Heraklion, UNINET in Oslo, and University of Pennsylvania in the US. In all cases we deployed a *SweetSpot* sensor and a NID. At the time of deployment the *Argos* sensor was still incomplete, so we decided to deploy it only at the Vrije Universiteit Amsterdam. In addition, the evaluation criteria of the *Argos* sensor are different from those of *SweetSpot*. For instance, what is crucial for *Argos* is whether it catches all attacks that use buffer overflows and whether or not we incur any false positives. For this reason, we evaluate *SweetSpot* and *Argos* separately. The prime goal of our evaluation is to prove the ability of *SweetBait* to generate valid worm signatures, and achieve a low reaction time.

The size of unused IP address space varied from 32 IPs in ICS FORTH to two class C subnets in Vrije Universiteit. As expected, the larger the address space, the more traffic was captured by the honeypot and consequently more signatures were generated. Additionally, we noted increased activity in our University of Pennsylvania honeypot, which may be caused by the higher density of IP addresses in this area. As we did not have access to a router to redirect suspicious traffic to our honeypots, the *SweetSpot* sensors currently only pick up random IP scanning worms.

A. *SweetSpot experiments*

Initially, we ran *SweetBait* with *SweetSpot* sensors for 24 hour lapses, to get a first glimpse of the generated signatures, and tune the system. We set up *SweetSpot* to emulate hosts running the following operating systems: Linux kernel 2.4.20-2.5.20, Windows XP Professional RC+1, and MS Windows Professional Advance Server Beta3. Additionally, the hosts emulated services such as FTP, POP3, and IIS application server, while accepting connections on all ports. Obviously, such a choice is not suggested for a production system, since it would expose the honeypot, but it is ideal for maximising the captured traffic during evaluation.

1) Signature Generation: Exceeding our expectations we collected a significant number of signatures in just a couple of hours. Using the values given in Section III-E, signature specialisation reduced the tens of thousands of signatures generated by *honeycomb* to tens. Table I depicts this for five of our experiments, while the cumulative number of new signatures found at the honeypot and the CC respectively is shown in logscale in Figure 3. The plot shows that the number of signatures in the CC is a small fraction of the total number of signatures generated in *SweetSpot*. Most are refinements or signatures of previous ones. *SweetBait* also generated signatures that could not be applied to a NIP sensor, because it would not be able to discriminate between legitimate and malicious traffic, resulting in the *whitelisted* signatures shown in Table II.

After applying the *whitelist* at *SweetSpot* the results were further improved. The generated signatures consisted of well-known older worms such as CodeRedII, Slammer, MSBlaster and Nimda [22], [23], [24], [25], as well as many exploit attempts including the more recent Veritas backup exec and Microsoft WINS [26], [27] vulnerabilities. A detailed list of all generated signatures can be found on-line at `http://www.few.vu.nl/signatures.html`. Some of the generated signatures seem peculiar, because of long repeated sequences in their payloads such as NOPs. Nevertheless, they are still usable and will not cause false positives, since no legitimate traffic would match the protocol, port number and content triplet.

A significant number of signatures is indirectly related to malicious network traffic, e.g., the signatures generated from traffic targeting backdoors created by worms on infected hosts (like MyDoom and Sasser [28], [29]). Additionally, many apparently benevolent messages, under closer inspection proved to be probes looking for active hosts and services. An example of such a message is a NetBIOS name service wildcard query, which precedes attacks on NetBIOS sessions service [30]. Even though these signatures are somewhat connected with malicious activity, they are not applicable, because they might also hinder access to legitimate users, and have thus been added to the *whitelist*. More such signatures were for instance generated when attackers attempted multiple connection attempts to MSSQL servers. Even though such attempts are obviously of a malicious nature, the resulting signatures cannot always be of practical use, since they might block access to public servers in the network. As discussed earlier, *SweetBait* offers various means to help ensure that the activity of such signatures does not rise high enough to cause their filtering.

Most of the signatures are less than 200 bytes long. Small signatures focus on the exploit used by a worm, and permit us to deploy more of them on the NID and NIP sensors. The distribution of the size of the generated signatures is shown in Figure 4. The fact that the length of most signatures is smaller than an IP packet does not imply that the worms used a single packet to propagate. The majority of the signatures involved the TCP protocol, and only Slammer's propagation and an SNMP attack were performed over UDP. To handle TCP fragmentation *honeycomb* employs flow reconstruction, and can identify patterns across multiple TCP packets.

In Section III-E we described how we quantify the virulence of each signature. While we only monitored the portion of the network traffic towards the honeypot, rather than the total network traffic, we were quite able to track the virulence of attacks. For example, in Figure 5 we plot the virulence for three of the most aggressive attacks.

2) Performance: To complete the evaluation of our system, we conducted measurements regarding the performance of the CC. It has to be able to process all the received information in a reasonable amount of time to achieve a low reaction time to worm outbreaks. Because of the nature of the honeypot sensor, the amount of traffic sent to the CC is negligible, while the number of alerts generated during an outbreak could be overwhelming.

We conducted experiments with a NID sensor generating false alerts to stress test the throughput of the CC, and locate possible bottlenecks. We set up the NID sensor to continuously transmit alerts, and measured the number of alerts that were processed every second at the CC. To achieve more realistic results, a honeypot sensor was also connected and was sending signatures. Initially, the NID sensor did not use alert caching, which resulted in a maximum throughput of just 15 alerts per second on average. Investigating the cause of such poor performance, we discovered that the database needed approximately 70 msec to store a single alert. Using faster hardware to host the *control centre* would definitely improve throughput, since we used a slow PC with 256MB of memory running at 1.2GHz. At any rates, switching to alert caching helped us overcome this limitation by achieving a throughput of approximately 140,000 alerts per second.

Another aspect of performance is the time it takes for the control centre to initiate monitoring, and consequently filter a new worm signature. As we have mentioned before, the generation of signatures depends on repeated byte patterns being identified by *honeycomb*. This implies that the speed at which a signature is generated,depends on the speed of the worm itself. As soon as the signature is generated, it is send to the CC. The time needed to initiate monitoring, depends solely on the period T that the CC updates NID and NIP sensors. Exchange with the GCC is also performed periodically, so

TABLE I

SPECIALISATION RESULTS

Usable signatures *(honeycomb log)*	**Unique signatures** *(honeycomb log)*	**CC entries** *(database)*
23400	12039	14
2861	439	9
6030	2107	11
35500	7462	20
43470	21957	21
323237	3538	27

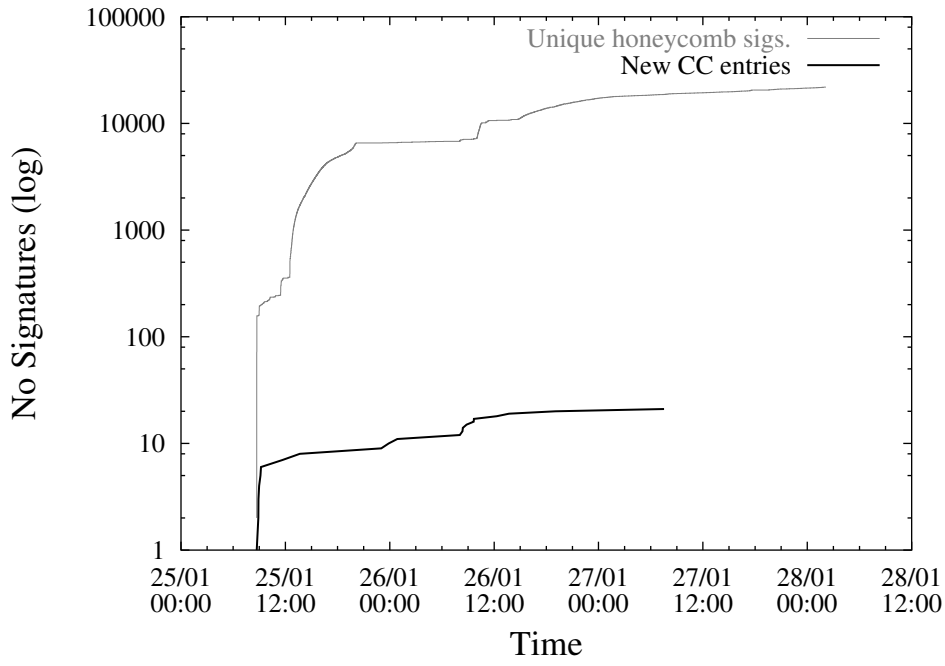

Fig. 3. Signature specialisation

TABLE II

WHITELIST

```
alert udp any any -> any 137 (msg: "NetBIOS Name Service Wildcard Query";
pcre: "\x00*\x0F*\x00*\x01\x00*\s*CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x00*!
\x00*\x01*\$"; )

alert tcp any any -> any any ( msg: "NULL packets"; pcre: "^\x00+$"; )

alert tcp any any -> any 80 ( msg: "HTTP Request(Used for DoS)"; pcre:
"^GET / HTTP/1\.1\r\nAccept\x3A image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, \*/\*\r\nUser-Agent: Mozilla/4\.0 (compatible; MSIE 5\.5;
Windows 98)\r\nHost\x3A .+\r\nConnection\x3A Keep-Alive\r\n\r\n$"; )

alert tcp any any -> any 139 ( msg: " \Session Request to SMBSERVER";
pcre: "\x81\x00\x00D [A-Z]{32}\x00 [A-Z]{32}\x00$"; )

alert tcp any any -> any 139 ( msg: "Session Request to SMBSERVER";
pcre: "A\x00 [A-Z]{32}"; )

alert tcp any any -> any 80 ( msg: "IIS WebDAV request"; pcre: "^OPTIONS
 / HTTP/1\.1\r\ntranslate\x3A f\r\nUser-Agent\x3A Microsoft-WebDAV-
MiniRedir/5\.1\.2600\r\nHost\x3A .+\r\nContent-Length\x3A 0\r\n
Connection\x3A Keep-Alive\r\n\r\n$" );
```

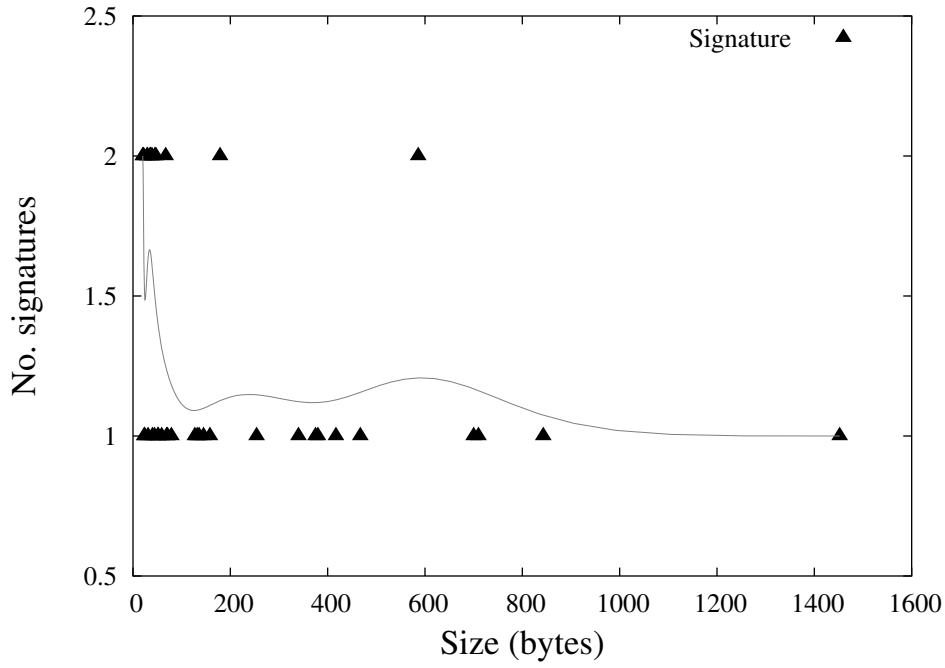

Fig. 4. Signature size distribution

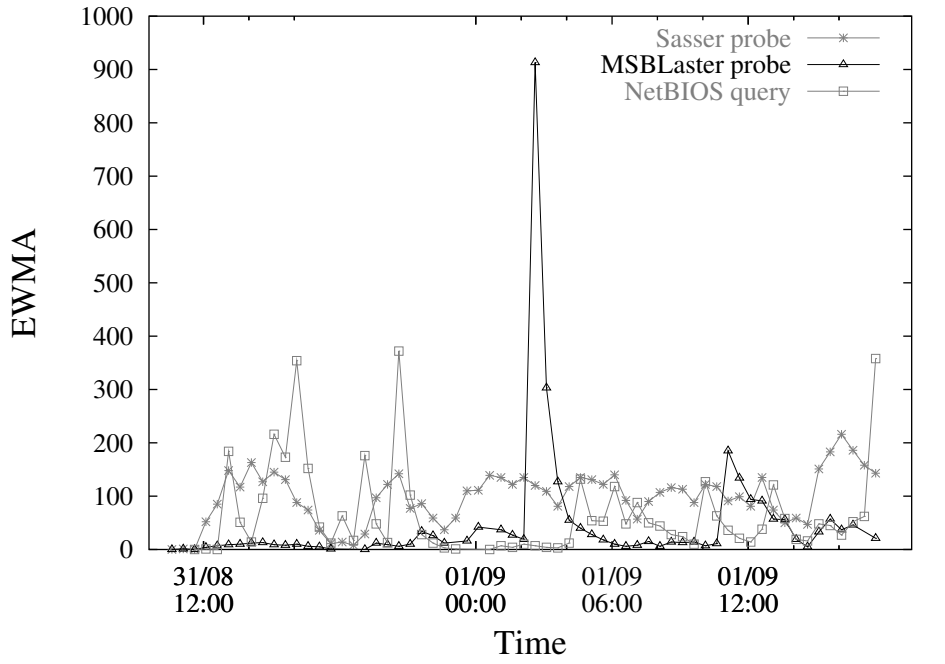

Fig. 5. Virulence: the exponentially weighted moving average (EWMA) is *SweetBait*'s measure of virulence. The plot shows for instance that MSBlaster was very active in the morning of 01/09.

the maximum time needed for a signature to be distributed globally is $2 \times T$, assuming all deployed systems have the same period. The same $2 \times T$ has to elapse as well, before filtering a signature, assuming that its activity has exceeded the defined threshold. To conclude, a short update period leads to fast reaction times against new worms, while the signature will be more refined in later updates. A distributed GCC would overcome potential scalability and availability issues for GCC, but is beyond the scope of this paper. During our evaluation, T was set to 2 minutes and we were pleased to observe that no unexpected performance degradation occurred.

B. Argos Evaluation

We ran *Argos* with Windows XP against a set of attacks present in the Metasploit framework that we could run without having to buy additional software [31]. The tests included different types of buffer overflow (off-by-one, heap, stack). For instance, some famous exploits included: `LSASS MS04-011 Overflow`, `PnP MS05-039 Overflow`, `ASN.1 Library Bitstring Heap Overflow`, and `RPC DCOM MS03-026`. No test incurred false negatives. In addition, we did not incur false positives while operating the system. As our experience with Linux is far more extensive than with Windows, we conducted additional homegrown tests with this OS, including various types of buffer overflow and format string attacks. Again, all were detected succesfully and we did not incur false positives.

The injection of our own shellcode in the process was also tested successfully in Linux. Currently, the shellcode extracts the identifier of the process and sends it via UDP to our signature generator. On Linux we completed a partial implementation of the signature generation of encrypted worms. An interposing function intercepts data going in and out of the `libssl` and `libcrypto` functions. For instance, by writing interposers for calls like `SSL_read()` and `BIO_read()` we were able to get at the data before it is sent to the application and test it for the presence of a pattern. To our knowledge we are the first to use this method for protection against encrypted worms.

Finally, we tested the performance of *Argos* in terms of overhead generated by the underlying emulation and instrumentation framework. All tests were conducted on a 2GHz AMD Athlon XP 2800 processor with 512K L2 cache and 1GB of RAM running Gentoo Linux with kernel 2.6.12.5. The emulated PC was a Pentium II™ with a 128K L2 cache and 512MB of memory. For optimal performance we did not use a file as a virtual hard disk, but instead dedicated a single IDE UDMA133 hard drive to the emulator. The guest OS used for the benchmark test was Slackware Linux 10.1 with kernel 2.4.29.

The performance overhead of *Argos* in terms of slowdown compared to native execution is shown in Table III. Two versions of Qemu were used, the original unmodified Qemu indicated as 'Vanilla Qemu', and secure Qemu which uses our memory tracking system. It should be mentioned that we have measured performance without the proprietary QEMU accelerator which speeds up QEMU to roughly half the performance of running directly on the hardware. The applications/benchmarks tested where bunzip 2 1.0.3, the httperf 0.8 web server benchmark, and BYTE magazine's Unix benchmark nbench 2.2.2.

We calibrated httperf for each platform separately to request the web server's main page 'index.html', so as to maximise the number of processed requests per second. Individual calibration was necessary, because Qemu's virtual network interface architecture introduces lag time that caused many HTTP requests to timeout when the same load as the native system was used. The metric adopted for comparison of web server performance was the number of milliseconds per request. The web server employed for the httperf benchmark was apache 2.0.4. The nbench produces a performance index for each platform's integer, float, and memory operations. This index specifies how the system compares with an AMD K6 at 233MHz with a 512K L2 cache, and it was used to compare the platforms.

TABLE III
Argos PERFORMANCE BENCHMARKS FIGURES

program	Vanilla Qemu	Secure Qemu
bunzip2	7.77	16.58
httperf	21.6	26.05
nbench integer	10.05	18.89
nbench float	21.06	25.48
nbench memory	12.39	21.48

The performance overhead of secure Qemu varies between a 16 times slowdown for bunzip2, and a 26 times slowdown for apache as reported by nbench. Even though the overhead is certainly not negligible, an OS running under secure Qemu is still able to function in sensible margins and could host multiple services. We emphasize that *Argos* is used as a honeypot rather than a production machine.

1) Signature Aliasing: One aspect of our system that we have not yet explored in detail is may be termed 'signature aliasing': the phenomenon that sensors of different type generate very different signatures for the same attack. For example, both *SweetSpot* and *Argos* generate a signature for the Slammer worm. Because the algorithms used for signature generation differ consiserably, these signatures will not be the same.

It may that aliasing is a good thing, as it increases the probability of catching a worm. However, multiple signatures of a virulent attack may also take up a disproportionate amount of a NID/HID/NIP sensor's signature budget. One possible solution is to extend *SweetBait* in such a way that signatures that always coincide are marked as equivalent, in which case only one will be activated. However, this is left for future work.

V. RELATED WORK

Much of the related work we already discussed in-line. In this section we highlight projects or aspects of projects that did not fit well in the main body of text. Most well-known systems (e.g., HayStack [32]) fall in a category of detection only and no active response. Some, however, do attempt automated response.

Various types of honeypots have been used for worm detection. A network of high-interaction honeypots is used to capture worms in the honeynet project [10]. By analysing network traffic and the honeypot's state it is possible to produce detailed descriptions of worm behaviour. Successful

application of low-interaction virtual honeypots was demonstrated by Laurent Oudot in capturing and counter-attacking the MSBlaster worm [33]. LaBrea [34], is a honeypot used as a tarpit: it slows down scanning worms, by keeping TCP connections open indefinitely. While effective for some worms, it would be powerless in the face of a UDP worm like Slammer. Sombria [35] is yet another honeypot system that has been setup for research purposes in Japan. All of these projects differ form *SweetBait* in that the focus is on capturing worms, rather than on automated response based on automatically extracted and refined signatures.

Honeycomb automatically creates signatures based on a longest common substring and has been successful in generating accurate signatures for the Slammer and Code Red II worms [7]. Nevertheless it can be fooled by long sequences of bytes repeated by certain protocols such as NetBIOS, creating signatures for otherwise legitimate traffic. Another system that automatically generates signatures for TCP worms is AutoGraph [36]. It operates by analysing prevalence of portions of flow payloads and exhibits a fairly low false positives rate. Like *SweetBait* it operates better in a distributed environment. However, it is not aimed at finding refinements or generalisation of signatures, nor is it currently coupled to an automated response system.

Joukov and Chiueh propose a worm containment environment that combines anomaly detection, egress filters and honeypots to generate worm signatures and filter them at the enterprise firewall [37]. Its major weakness is that even unsophisticated polymorphic worms may be able to circumvent detection.

A similar system also addressing the issue of an Internet wide centre to correlate warnings and share information is described by Changchun Zou et al. [38]. Ingress and egress scan monitors are distributed in different parts of the network and submit their warnings to a *malware* warning centre. The monitors are using a Kalman filter to identify the propagation of a worm based on observed illegitimated scan traffic. The approach aims to detect zero-day worms at their early stage, but is vulnerable to background noise that could cause a high rate of false alarms.

EarlyBird is another system that aims to fingerprint worms at an early stage [39]. It scans payloads and correlates the information with a set of unique addresses of sources that are spreading the worm and destinations under attack. In-band content inspection is also the goal of [40]. In this case, however, the authors have pushed the firewall all the way to the end-node and implemented the IDS in hardware on the NIC. Neither of the latter two approaches copes well with polymorphic worms. In *SweetBait*, we are currently adding our own in-band content inspection based on FFPF and Intel network processors [41].

A fast containment system is presented in [42]. It differs from most projects described here, including ours, in that it takes a network-centric view and aims at implementation in hardware.

A cooperative immunisation system against worms is described by Anagnostakis et al. [43]. The system consists of distributed nodes exchanging information about threats and appropriate counter-measures. It is based on scanning incoming traffic for malware at the hosts and like *SweetBait*, it determines which signatures to scan for based on observed virulence. The system is a network-centric approach that bears some resemblance to *SweetBait*'s honeypot approach. As trust is based on validating information by asking multiple nodes, the system becomes vulnerable if a large number of nodes is compromised.

A more aggressive approach is adopted by Sidiroglou and Keromytis [44]. Like *SweetBait* they deploy diverse sensors including network monitors and honeypots. The honeypots used are highly-interactive, running real versions of popular applications to be protected. The applications run in a sandbox environment. They are monitored for illegal behaviour, and when such behaviour is detected the error that caused it is located and a patch is automatically generated and distributed through a software update service. The risk of such active measures is that an automatically generated patch could do more harm than good and it leaves the possibility of gaming by hackers; carefully crafted input to the honeypots could cause the generation of patches that create weaknesses.

Honeystat protects against scanning worms by employing high-interaction honeypots. Once a host is compromised it monitors CPU, memory, network and disk events to capture the behaviour of the worm [6]. It produces accurate signatures, but has no method of refining the signatures in the way provided by *SweetBait*.

Besides Honeypots there are various other approaches to intrusion detection and prevention. Anomaly detection systems (ADS) are able to detect zero-day worms and may even work at high speeds [45], [46], [47], [48]. Unfortunately, they tend to be fairly inaccurate and are commonly tuned conservatively to keep the number of false positives low.

Some systems are made intrusion-tolerant and use application diversity to compare outcomes of different implementations when it is suspected that a server has been compromised [49].

A well-known IDS is Vern Paxson's Bro [50]. Compared to *SweetBait*, Bro gives more attention to event handling and policy implementation. On the other hand it counts over 27.000 lines of C++ code, is implemented for instance on top of `libpcap` and may not assume that all traffic is suspect. Altogether this makes it a very different approach.

Argos is related to Minos which uses a similar form of memory tainting to detect buffer overflows [51]. Unlike *Argos*, Minos does not generate signatures. Moreover, it aims to be used in hardware in new processor designs. For this reason it uses BOCHS rather than an emulator. As there no such hardware is currently available, the performance of Minos is almost an order of magnitude worse than that of *Argos*.

VI. Conclusions and Future Work

In this paper we discussed the design and implementation of *SweetBait*, a system that is a combination of network intrusion

detection and prevention techniques. It employs different types of honeypot sensors, both high-interaction and low-interaction. It was shown that *SweetBait* is able to automatically generate signatures for random IP address space scanning worms without any prior knowledge. For non-scanning worms, we have shown a solution in which we advertise a high-interaction honeypot that uses memory tainting to detect buffer overflows and automatically generates a signature by correlating the memory footprint with network traces. We also demonstrated how this information can be distributed and deployed without any human intervention minimising reaction time to zero-day worms. Furthermore, the signature specialisation, activity prediction and automatic deployment techniques introduced provide a valuable administration tool, which condenses the information that needs auditing by administrators, while self-adapting to ensure a high throughput of the monitoring nodes.

Our future plans to improve the high-interaction honeypots signature generation. We believe that given the wealth of information about the attack, we should be able to generate better and more detailed signatures both for HID/NID systems and human experts. We also plan to further investigate 'signature aliasing' where different signatures correspond to the same attack.

ACKNOWLEDGMENTS

This work was sponsored in part by the EU FP6 NoAH project and the Dutch NWO Deworm project. The authors would like to thank Kaiming Huang of Xiamen University in China for his contribution to *cardguard* and Asia Slowinska of the Vrije Universiteit Amsterdam for her work on TCP stream reconstruction and signature detection. We are grateful to Intel for providing us with a set of Intel IXP1200 boards which were used for the *cardguard* HIP. Finally, a massive thanks is owed to ICS-FORTH, the University of Pennsylvania and UNINETT for hosting the *SweetBait* system.

REFERENCES

[1] Stuart Staniford, Vern Paxson and Nicolas Weaver, "How to 0wn the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, 2002.

[2] S. Staniford, D. Moore, V. Paxson, and N. Weaver, "The top speed of flash worms," in *Proc. of the 2004 ACM Workshop on Rapid malcode (WORM'04)*. New York, NY, USA: ACM Press, 2004, pp. 33–42.

[3] C. Systems, "Cisco secure intrusion detection system version 2.2.0 user guide (netranger)," 2003.

[4] N. Provos, "A virtual honeypot framework," CITI, Tech. Rep. 03-1, 2003.

[5] M. Costa, J. Crowcroft, M. Castro, and A. Rowstron, "Can we contain internet worms?" in *Third Workshop on Hot Topics in Networks (HOTNETS-III)*, San Diego, CA, November 2004.

[6] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen, "HoneyStat: Local worm detection using honeypots," in *Proc. of RAID2004*, Sophia Antipolis, France, September 2004.

[7] C. Kreibich and J. Crowcroft, "Honeycomb - Creating Intrusion Detection Signatures Using Honeypots," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 51–56, January 2004.

[8] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-end containment of internet worms," in *SOSP'05*, Brighton, UK, October 2005.

[9] H. Bos and K. Huang, "Towards software-based signature detection for intrusion prevention on the network card," in *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, WA, September 2005. [Online]. Available: http://www.cs.vu.nl/~herbertb/papers/cardguard_raid05.pdf

[10] J. Levine, J. Grizzard, and H. Owen, "Using honeynets to protect large enterprise networks," *IEEE Security & Privacy*, vol. 2, no. 6, pp. 73–75, November/December 2004.

[11] X. Jiang and D. Xu, "Collapsar: A vm-based architecture for network attack detention center." in *USENIX Security Symposium*. USENIX, 2004, pp. 15–28.

[12] N. Provos, "A virtual honeypot framework," in *13th USENIX Security Symposium*, San Diego, CA, August 2004.

[13] N. Vanderavero, X. Brouckaert, O. Bonaventure, and B. L. Charlier, "The honeytank: a scalable approach to collect malicious Internet traffic," in *Proc. of IISW04*, December 2004.

[14] M. Dornseif, T. Holz, and C. Klein, "Nosebreak - attacking honeynets," in *Proceedings of the 5th Annual IEEE Information Assurance Workshop*, 2004. [Online]. Available: http://md.hudora.de/publications/2004-NoSEBrEaK.pdf

[15] Ofir Arkin and Fyodor Yarochkin, "Xprobe v2.0: A "Fuzzy" Approach to Remote Active Operating Systems Fingerprinting," http://www.xprobe2.org, August 2002.

[16] Fyodor Yarochkin, "Remote OS Detection via TCP/IP Stack Fingerprinting," http://www.nmap.prg/nmap/nmap-fingerprinting-article.html, October 1998.

[17] F. Bellard, "Qemu, a fast and portable dynamic translator," in *USENIX 2005 Annual Technical Conference, FREENIX Track*, Anaheim, CA, April 2005, pp. 41–46.

[18] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," in *Proc. of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

[19] M. Roesch, "Snort Lightweight Intrusion Detection for Networks," in *Proceedings of USENIX LISA '99: 13th Systems Administration Conference*, 1999.

[20] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," in *Communications of the ACM*, G. Manacher, Ed., vol. 18, June 1975.

[21] M. Dacier, F. Pouget, and H. Debar, "Honeypots: Practical means to validate malicious fault assumptions," in *10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2004)*, Papeete, Tahiti, March 2004.

[22] Symantec, "CodeRedII," http://www.symantec.com/avcenter/venc/data/codered.ii.html, August 2001.

[23] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford and Nicholas Weaver, "Inside the slammer worm," *IEEE Security & Privacy*, July/August 2003.

[24] Symantec, "w32.blaster.worm," http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.wor%m.html, August 2003.

[25] CERT, "CERT Advisory CA-2001-26 Nimda Worm," http://www.cert.org/advisories/CA-2001-26.html, September 2001.

[26] SecuriTeam, "Veritas backup exec agent browser registration request exploit," http://www.securiteam.com/exploits/5ZP0G0KEKQ.html, January 2005.

[27] K-Otik, "Microsoft wins remote code execution exploit," http://www.k-otik.com/exploits/20041231.ZUC-WINShit.c.php, December 2004.

[28] Symantec, "W32.mydoom.a@mm," http://securityresponse.symantec.com/avcenter/venc/data/w32.novarg.a@mm%.html, January 2004.

[29] ——, "w32.sasser.worm," http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm%.html, 2004.

[30] M. Ligh, "Attack signatures and internet traffic analysis," http://www.mnin.org/forums/viewtopic.php?t=73, 2004.

[31] M. Project, "Metasploit Framework," http://www.metasploit.com/.

[32] S. E. Smaha, "Haystack: An intrusion detection system," in *IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, USA, December 1988.

[33] L. Oudot, "Fighting internet worms with honeypots," http://www.securityfocus.com/infocus/1740, October 2003.

[34] T. Liston, "Welcome To My Tarpit: The Tactical and Strategic Use of LaBrea," http://www.threenorth.com/LaBrea/LaBrea.txt, 2001.

[35] S. S. Service, "A Walk Through "Sombria": A Network Surveillance System," http://www.lac.co.jp/business/sns/intelligence/sombria_e/smbr_1.pdf, July 2003.

[36] H.-A. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," in *Proc. of the 13th USENIX Security Symposium*, San Diego, CA, August 2004.
[37] N. Joukov and T. cker Chiueh, "Internet worms as internet-wide threat," Experimental Computer Systems Lab, Tech. Rep. TR-143, September 2003.
[38] Cliff Changchun Zou, Lixin Gao, Welbo Gong and Don Townsley, "Monitoring and early warning for internet worms," in *Proceedings of the 10th ACM conference on Computer and communication security*, 2003, pp. 190–199.
[39] S. Singh, C. Estan, G. Varghese, and S. Savage, "The EarlyBird system for real-time detection of unknown worms," in *Operating System Design and Implementation (OSDI)*, San Francisco, Ca, December 2004.
[40] C. Clark, W. Lee, D. Schimmel, D. Contis, M. Koné, and A. Thomas, "A hardware platform for network intrusion detection and prevention," in *Third Workshop on Network Processors and Applications*, Madrid, Spain, February 2004.
[41] H. Bos, W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis, "FFPF: Fairly Fast Packet Filters," in *Proceedings of OSDI'04*, San Francisco, CA, December 2004.
[42] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *13th USENIX Security Symposium*, San Diego, August 2004, pp. 29–44.
[43] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis and D. Li, "A Cooperative Immunization System for an Untrusting Internet," in *Proceedings of the 11th IEEE Internation Conference on Networking (ICON)*, September/October 2003.
[44] S. Sidiroglou and A. D. Keromytis, "A network worm vaccine architecture," in *12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2003.
[45] T. Toth and C. Kruegel, "Connection-history based anomaly detection," in *Proc. of the IEEE Workshop on Information Assurance and Security*, West Point, NY, Jun 2002. [Online]. Available: citeseer.ist.psu.edu/toth02connectionhistory.html
[46] K. Wang and S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection," in *Proc. of RAID2004*, Sophia Antipolis, France, September 2004.
[47] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proc. of ACM CCS*, Washington, D.C., October 2003, pp. 251–261.
[48] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford, R. Yip, and D. Zerkle, "The design of GrIDS: A graph-based intrusion detection system," UC Davis, Tech. Rep. CSE-99-2, January 1999.
[49] J. E. Just, J. C. Reynolds, L. A. Clough, M. Danforth, K. N. Levitt, R. Maglich, and J. Rowe, "Learning unknown attacks - a start." in *RAID*, 2002, pp. 158–176.
[50] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31(23-24), pp. 2435–2463, December 1999.
[51] J. R. Crandall and F. T. Chong, "Minos: Control data attack prevention orthogonal to memory model," in *Proc. of the 37th annual International Symposium on Microarchitec ture*, 2004, pp. 221–232.