Hybrid Intrusion detection network monitoring with honeypots

Jako Fritz

April 9, 2011

Contents

1 Introduction **3**
- 1.1 Hybrid detection . . . 7
- 1.2 Research Questions . . . 9

2 Literature **10**
- 2.1 Malware taxonomy . . . 10
- 2.2 Host-based detection . . . 12
  - 2.2.1 Low-interaction honeypot . . . 14
  - 2.2.2 High interaction honeypot . . . 15
  - 2.2.3 Summary . . . 18
- 2.3 Network-based detection . . . 21
  - 2.3.1 Signature-based approaches . . . 22
  - 2.3.2 Anomaly-based approaches . . . 25
  - 2.3.3 Botnet detectors . . . 27
  - 2.3.4 Other . . . 29
- 2.4 Network statistics and containment strategies . . . 31
- 2.5 Conclusion . . . 32

3 Hybrid architecture **35**
- 3.1 Design Goals . . . 37
- 3.2 Functional description . . . 39
  - 3.2.1 Network traffic capture . . . 39
  - 3.2.2 Traffic redirector component . . . 42
  - 3.2.3 First stage detectors Network-based anomaly detectors . . . 43
  - 3.2.4 Second stage detectors . . . 44
  - 3.2.5 Black hole . . . 45
  - 3.2.6 Manager . . . 45
- 3.3 Detection strategy . . . 46
  - 3.3.1 Initiating monitoring . . . 47
  - 3.3.2 Declaring false positives . . . 47
  - 3.3.3 Selecting second stage detectors . . . 47
  - 3.3.4 Sensor placement and Mac spoofing . . . 52

4 Evaluation **54**
4.1 Introduction 54
4.2 Prototype implementation 55
4.2.1 Sandbox 55
4.2.2 Hybrid Detector 56
4.2.3 Traffic capture 60
4.3 Experiments 61
4.3.1 Limitations of the experiment 63
4.3.2 Comparison with single honeypot 64
4.4 Analysis of results 64

5 Conclusions **68**
5.1 Revisit research questions 68
5.1.1 Honeypots as network detectors 69
5.1.2 How does it compare 69
5.1.3 Suitable detectors 69
5.1.4 Making errors 70
5.1.5 Future research 71
5.2 Conclusion 72

Chapter 1

Introduction

The malware problem Malware is (malicious) code that is unwanted and therefore is spread (or will spread itself) by any means possible while attempting to hide from existing detection mechanisms. Unfortunately malware is often successful in avoiding detection and security systems still struggle to prevent infections by malware from occurring, or even detecting that an infection has taken place. Of particular concern is malware that spreads itself by exploiting some software vulnerability as it can not be stopped by educating users, nor as noticeable as malware that propagates purely by user based interactions via phishing or scare tactics.

The most well known instances of autonomous malware which directly attacks other nodes via their network services are Internet worms. Preventing or containing infection by worms is something which is quite difficult to achieve as can be seen by the persistent and continued presence of the Downadub/Conficker worm and more recent the Stuxnet and the 'On the Move' email worm. The latter spreads by emailing itself to email addresses present in an address list, but requires a user to click on or execute the attachment, the other two are fully automated. The Downadup worm was released in the fall of 2008, and was still causing significant trouble at the beginning of 2010, almost 18 months after its initial release. Stuxnets claim to fame is that it is malware that is (assumed to be) designed to disrupt specific critical infrastructure (such as nuclear or hydro power plants) and which exhibits evidence of being engineered by a well funded team of professionals. It serves as a stark reminder how ill quipped current systems and protection measures are at detecting and dealing with malware.

Malware generated network traffic Malware is designed to perform one or more functions. In most cases, at least one of these functions involves the use of the network. Examples are functions that spread a binary file containing the malware or facilitate the uploading of the binary to other machines, sending Spam, running proxy services for various purposes, stealing of user credentials and snooping on network traffic on the infected node. Constants

of the malware life cycle are that malware must be distributed or distribute itself, and that it must perform a malicious action at least once. This means that determining whether a node is infected with malware can be achieved by monitoring for the presence of traffic generated by these malicious functions. However malware detection via network-based analysis is a thorough task requiring invasive network analysis which takes either too many resources or lacking accuracy.

example attack A memory corruption attack aims to smash through host-based protective measures and inject some custom crafted code into a vulnerable (remote) process. The injected code can for instance be used to gain control of the attacked system. Such an attack consists of a string of bytes, commonly referred to as exploit code, which is injected into a target process and which contains random filler data, self-contained attack code called shell code, some glue code that diverts the execution code from the attacked process to the crafted attack code and optionally some data that is fed to the attacked process to ensure it is in the right state for exploitation. For example the aforementioned Internet worms utilize, amongst others, this type of attack to execute malicious code which ensures an instance of the malware is executed on an attacked node. A classic example of a memory corruption attack string, a stack based buffer overflow exploit, is depicted in Figure 1.1.

Buffer overflow The attack depicted is a stack based buffer overflow where due to some error more data is copied onto the stack then the program allocated space for. This typically results in the excess data overwriting data previously stored on the stack such as various pointers and function arguments. Since the stack also holds the instruction pointers used to resume execution when program functions return it becomes possible to arbitrarily select from what memory address code execution should resume

How the attack works The filler buffer and sEIP value together ensure that the targeted application is in the right state to be successfully exploited. The sEIP value diverts the flow of execution by changing the instruction pointer stored on the stack to the value of sEIP. Once the exploited function returns the malicious code is executed. The NOP sled and shellcode represent the self contained attack code. The NOP sled is required because memory locations of items on the stack are not fixed so the location of shellcode on the stack can only be approximated. A NOP sled is prefixed to the actual shellcode such that the value of sEIP will point to somewhere in between the beginning and the end of the NOP sled. Starting from the sEIP address the NOP instructions will be executed until we slide into the shellcode, hence the name.

Detecting malicious traffic using Network Intrusion Detection Rule-based network detection systems aim to detect instances of this attack by matching network traffic with a description of this attack, which is called a signature. Signatures are constructed by reducing sampled attack strings to a few key sequence of bytes that are specific to these attack strings and not to any other network traffic. However the values used in these attack strings are coupled to the nature of the vulnerability. If this coupling is tight there may only be one possible sequence of bytes leading to a successful attack. This is typically not the case which reduces the monitoring for attacks on a particular vulnerability to monitoring for any known exploits targeting that particular software vulnerability. As the number of vulnerabilities and exploitation techniques grow this requires recreating, tuning and applying more signatures. Signatures for the aforementioned attack would focus on the presence of any combination of the used shellcode, the presence of a NOP sled, the memory value sEIP and any fixed byte sequences in the filler data.

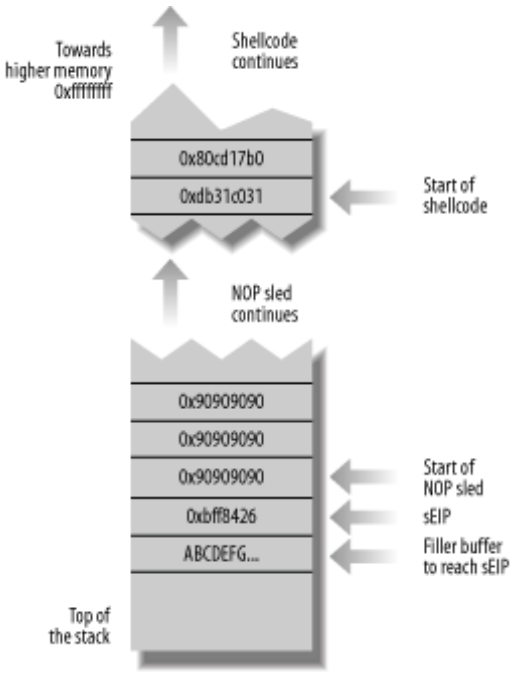

Figure 1.1: Source:Network Security Tools By Justin Clarke, Nitesh Dhanjani

A different path towards network detection is taken by anomaly-based detectors. These type of detectors focus on features of network traffic and the variance between these features, if an observed feature is different from the features that are normally observed this is an anomaly and could be malicious. Examples are approaches such as Payl/Anagram [50],[36] which analyse byte frequencies in network traffic. The attack string might be injected into a variable that normally hold text data. When comparing the byte frequencies of normal text strings and the example attack string will show a large difference.

However network analysis alone is not sufficient to detect malicious traffic in time to prevent infection or even to detect that an infection may have taken place. Network analysis approaches require too much tuning, a priori information in the form of samples of malware or time to tune sensors to the point where the ratio of false positives/negatives becomes acceptable. Tuning is in itself no easy feat, as without additional aid it is not immediately obvious

what behaviour flagged by a network detector is actually malicious. Network analysis is not a useless endeavour, far from it, but high rates of detection or complete detection are typically not coupled with low false positive rates. To this rule some exceptions exist, but these typically focus on particular features present in only a subset of malware and so are not easily extended into generic detectors. This means in practice there is not a generic detector that can be employed, but instead multiple detectors are used that cover the spectrum of malicious behaviour.

Detecting malicious traffic using Honeypots A different approach are honeypots, which serve to capture malware samples and malicious behaviours. To achieve this honeypots consist of (or emulate) vulnerable systems, which are monitored in a variety of ways for changes made to the system. Since honeypots serve no other function there is no reason for a real user to interact with a honeypot. Any interaction therefore has a high probability of being malicious. It also turns out that monitoring for changes in honeypot is relatively easy to achieve. This makes honeypots very useful for detecting malicious behaviour and has lead to a variety of implementations which cover every conceivable malicious interaction or propagation method that malware can use. Honeypots approaches are limited to detecting malicious traffic that effects a change to the detector that is detrimental or defined to be bad behaviour, i.e. traffic that contains attacks or abusive behaviour. Detectors exists for detecting malicious traffic containing memory corruption and web based attacks, brute forcing of various login credentials for services and Spam.

The attack we illustrated earlier is relatively easy to detect by a honeypot as it smashes the stack and injects shellcode which under normal conditions should not be occurring. Off course before such an event occurs the attack code must first be injected into a vulnerable process, which requires an attacker to contact the honeypot. Furthermore to allow an attacker to execute this attack technique either a real or emulated vulnerable service must be present on the honeypot. The values used in the attack imply that this attack can only succeed on a Intel x86 32bit based Unix operating systems because it requires the process stack to reside in a specific memory region.

Honeypot limitations Honeypots can only interact with traffic sent to the IP addresses these are monitoring on, which turns out to be enough to track worm outbreaks on the Internet as demonstrated by the Honeystat system by Dagon et al [12]. However honeypots are not so efficient when the goal is to detect individual malware infections within a local network, this because observing an attack from a specific infected local node is much lower then the chance of observing an attack by any network connected node.

A second issue that hampers honeypots in detecting infected nodes is that any malicious action taken by such nodes must succeed, i.e. effect a measurable

change on a honeypot. Therefore the honeypot must be of such a type which facilitates or induces a malware program to perform its malicious action on. This issue is evident in memory corruption attacks which are tightly coupled to specific operating systems and individual patch levels. But also attacks that use loose security policies or abuse services require a specific detector. For example, detecting brute forcing of credentials requires a honeypot with a sufficient service implementation as to allow initial authentication exchanges to take place. For detecting attacks on applications provided by network services, typically web services on top of an HTTP server, some form of web application honeypot is needed. To predict what kind of detector is of the right type we can make use of available data on exploited vulnerabilities. In most cases attacks and abuse gravitate to particular operating systems and or services. This means that malicious attacks on a web server are likely to consists of various script injection techniques and not of memory corruption attacks. It is wrong to conclude that these attacks do not occur at all, but the probability of these occurring is considerably lower[61][1].

Finally honeypots are not infallible, malicious activity can go undetected if the detector focuses on specific aspects that a malicious interaction manages to avoid. An approach that monitors for code injection based on taint analysis may fail to detect malicious program uploads to its network storage shares. Changes to to the honeypot as a result of backscatter traffic from the network and operating systems creating changes may also trigger the detector component of the honeypot and cause false positives, but this does not appear to be a serious recurring issue in honeypot approaches.

1.1 Hybrid detection

The goal is to detect malware infected nodes on the local network by monitoring and rerouting the network traffic from these nodes. By intercepting traffic the malware is no longer effective as it can not abuse or attack other nodes. Our initial focus will be on detecting malware by monitoring for attacks or abusive behaviour as these traits are typically exhibited by malware and can also be monitored for using both network analysis and honeypots.

A hybrid detection system combines network-based detection and honeypots by redirecting network traffic to honeypot detectors using network redirection and traffic rewriting techniques. Network alerts are used as indicators to further investigate a node and determine what traffic is to be monitored using honeypots. This monitoring is achieved by redirecting some or all traffic from a node under investigation into honeypot acting as detectors of malicious attacks

[1]The reference is decidedly poor, but having trouble finding a paper on an empirical statistical study on exploited vulnerabilities (most focus on published vulnerabilities instead).

or abuse. The approach is aimed at detecting malware infections on (infected) nodes on a local network, so when we refer to malicious traffic we mean any traffic generated by malware originating from or flowing to an (infected) node on a local network. It is important to note that redirecting traffic is not performed on the fly, i.e. the traffic triggering the network-based detector is itself never rerouted to a honeypot, instead subsequent traffic originating from a node is redirected. This seems counter-intuitive, but its not our aim to corroborate whether an alert was valid for the specific traffic a network detector observed but whether this alert corresponds to an (infected) node launching attacks on other systems.

This detection in stages is why we label detectors that monitor traffic before redirecting as first stage detectors and honeypots, or any other applicable detector with similar properties, as second stage detectors. With second stage detector we mean a detector that interacts with traffic, and that provides network services which are monitored for attacks or abusive behaviour. The way detection is achieved should leave little room for errors, which is typically the case with honeypot detectors or spam traps. We sometimes use the more generic notion of second stage detectors instead of honeypots to refer to the detectors as not all detectors are true honeypots, and because the name implies a passive detector, where in the hybrid system we are actively routing traffic into honeypots and other applicable detectors. In that respect the detectors are more akin to classifiers of malicious behaviour.

Hybrid example The Threshold Random Walk (TRW) algorithm is an network-based detector aimed at detecting scanning behaviour exhibited by a node in the network. It monitors in an inexpensive way in terms of computation and storage requirements making it suitable for high speed or high volume network analysis. It operates on the basic premise that there exists a distinct (fixed) ratio of failed and successful TCP/IP connections and that this ratio is different for normal nodes and nodes that are scanning for vulnerable services. The TRW algorithm continuously computes a value using the number of failed or successful TCP connections as input values. For each node this computed value will (eventually) converge to some upper or lower boundary based on whether it exhibits scanning behaviour or not. Scanning behaviour can be observed because of many reasons, not all of them malicious, but we assume a machine that is scanning is looking for a another node to exploit, and thus we redirect the scanning traffic to a honeypot for a second stage detector. Should the scanning node indeed be looking for targets to exploit it is likely that it will decide that it has found a target and try, and hopefully succeed, in exploiting the honeypot. The event of attacking the second stage detector provides the proof that the scanning behaviour was indeed part of an attack. By monitoring in two stages using a network detector such as TRW it becomes largely irrelevant how the malware is launching attacks or what its attack

signature looks like. Instead the only required conditions are that the first stage detector can monitor for the divergence of normal behaviour, and that the second stage is capable of detecting the attacks launched or abusive behaviour exhibited by the malware.

1.2 Research Questions

- Given that honeypots are near perfect classifiers of malicious behaviour can honeypots be extended to have a larger network view by actively searching for and rerouting of malicious traffic to honeypot detectors?
- Rerouting traffic into honeypot detectors has a detrimental effect on network performance as a whole because also non malicious traffic may be affected, can we formulate a strategy that minimizes routing non-malicious traffic into honeypots?

In this thesis we attempt to answer both research questions by formulating an architecture that combines both network based detectors and host based detectors like honeypots and evaluate its performance using live malware in a controlled environment.

Chapter 2

Literature

Since a hybrid approach incorporates both network-based and host-based detection we will investigate the state of the art of both fields.We split our discussion of the current state of the art approaches into (1) detection of malware that spreads by abusing network services into host-based and network-based approaches and we (2) also look into research on malware traffic characteristics and worm containment strategies.

2.1 Malware taxonomy

Before discussing ways of detecting (automated) malware that attacks network services it is useful to understand how these differ from other types of malware. Malware typically is a piece of software engineered such that it can be used to gather information, launch network attacks, or perform other nefarious tasks like sending spam.

We distinguish two generic ways of distributing malware :

- Automatic distribution. Malware that distributes itself to other machines without requiring interference from users or the operators that deployed the malware. It may be the case that the malware relies on existing infrastructure to propagate, such as fixed locations to download malware binaries from. Automated propagation is in most cases achieved by either using an exploit for a vulnerable network service or program which subsequently downloads the malware to the target machine, or configuration errors that allow remote exploitation. Examples of the first are Internet worms and malicious web pages that target web browsers or programs used by web browsers, known as drive-by downloads. Examples of the latter category are poorly secured windows shares or USB sticks that are used to distribute malware.

- Manual distribution. Malware is merely software, but with a nefarious purpose, and it can be installed just like any normal program. This can

be achieved by installing malware directly or by attaching the malware to existing binaries, the latter is an example of the traditional computer virus. Asking users to install a particular program is a practical and simple approach for distribution and remarkably effective. Two common ways to trick users into installing malware is by either embedding the malware in a web page or by attaching it to an email and asking the user to install the program while promising some functionality. Because of attempts to educate users to not install programs distributed in this way recently a new approach is used which involves embedding malware into apparently normal programs and selling these to users as to give the appearance of it being legit software. The most well known examples of this type of approach are the various fake Anti Virus programs which pass themselves off as legit software, but really are malware.

Aside from the method used for distributing malware we can also differentiate based on the way malware is controlled and the method of propagation. The reason for classifying malware in this way is that when monitoring for malware it is the statistical divergence of its behaviour from the norm, and the way it interacts with a service both properties which govern the detection strategy and detectors that can be used.

Some malware will merely perform the function it was designed to do, such as collect information or facilitate distributing more copies of itself, whereas other malware is requires active control by the malware operators. For instance:

- Autonomous malware such as computer worms typically operate autonomously, distributing themselves to as many different machines as possible whilst and optionally may include some malicious payload.

- Non-autonomous malware on the other hand comprises malware that essentially functions as a server to which commands can be sent by an attacker, when this utilizes some sort of communication channel to control multiple instances of malware at the same time this is typically referred to as a botnet. The binaries to construct these botnets can be distributed manually or automatically, for example by using worms to install the botnet binary or instructing the botnet itself to target other nodes. In the latter case the observed network traffic is similar to that of worms, scanning for nodes running vulnerable services, and exploiting these to disseminate the botnet binary. These binaries however still are largely automated to execute a specific set of tasks, such as brute forcing login credentials or sending spam or other nefarious tasks that require the same set of actions over and over again, and typically are controlled via a unified communication channel allowing the control of many malware instances.

- manual malware, the classic backdoor program that allows an attacker full access to the system comes in many forms, and may even be part of malware that also incorporates features of botnet malware, but the important thing is that this program allows to execute actions on the victims machine and does not exhibit the automation features present in botnet malware. Although at this point the lines get rather blurry, as its perfectly viable to backdoor a remote program and leverage that access to setup a spam machines. The distinguishing feature here is no unified communication channel or repeated automated behaviour.

The distinction in the way these malware programs are controlled also affects how these are spread and the manor in which these manage to infect other nodes, worms typically attack services directly while non-autonomous malware will typically require an intermediate service provided by some other malware program to spread the malware such via email, web services, worms or peer to peer networks.

A generic way of grouping malware behaviour by the way it utilizes network services is

- Malware that directly abuse services on other nodes on a network via their network services, for example by abusing vulnerabilities, brute forcing login credentials or taking advantage of loose security policies causing it to be executed on the victims machine.

- Malware that uses other services to perform its tasks and attack other nodes, for instance by sending spam or trying to inject malicious URLs or code in websites, which in turn can attack other machines who visit those websites.

We focus on these attributes because these describe the types of malware that can be detected using a hybrid approach, that is to say in order to be detectable malware with a honeypot it must either directly attack a honeypot, or attack a service it provides with a malicious payload that indirectly targets victims. Furthermore because the hybrid approach stages detectors one after the other it is also imperative that some form of automation be present in the malware, it must exhibit repeated behaviour that first triggers a network detector and again to be detected at the second stage detector. Finally malware needs to employ some form of automation in how it spreads, for example malware that uses scareware or fishing tactics to get installed onto a victims machine can not be detected by a hybrid detector.

2.2 Host-based detection

Host-based (malware) detection approaches are systems that monitor specific IP addresses and which may additionally provide (emulated) services to give

the appearance of a fully functional node or even an entire network. These approaches are commonly referred to as Honeypots.

Honeypots are specifically designed to detect malicious traffic that exploits network services or to study attacker behaviour. Because honeypots are not used for any other functions all traffic to a honeypot is considered to be either malicious traffic or background radiation[1]. This facilitates analysis of traffic, making it easier to observe scans and attacks since these need not be separated from non-attack traffic. But this is also the main limitation of a honeypot, it does not actively attract attack traffic but merely sits and waits until it is scanned or attacked.

Some relevant properties of honeypots are :

- Automation, The degree of automation of the honeypot, or how much manual interference is required when operating it.
- Detection capabilities, The detection capabilities of a honeypot indicate how well it is able to detect malicious activity and capture attack details. This also includes the accuracy of detections, particularly some high interaction honeypot approaches only monitor for specific actions or events leaving them somewhat vulnerable to malware that avoids exhibiting features the honeypot monitors for. Also the type of a detection, i.e. the class of malicious behaviour observed can vary, we focus on memory corruption vulnerabilities, but its also possible to construct a honeypot that for instance monitors for SQL injection attacks[2].
- Risk & Reliability, The risk of a compromise of the honeypot is always present, and can lead to serious damage if not properly contained. The way a honeypot is constructed and how it makes a detection affects the reliability of honeypot.
- Scalability, The resource usage of a honeypot governs many IP addresses the solution can monitor, which in turn determines how likely it is to capture an attack. A way to subdivide honeypots is to classify them as high interaction honeypots depending when these are constructed by using a full OS or low interaction when just parts of an OS are emulated. Typically a high interaction honeypot implements a detector using a complete and optionally weakened OS augmented with additionally services which may also be weakened and aggressive monitoring of the OS. Because of the high risk and the high degree of automation or similarity between

[1]Broadcast traffic is a generic designation for any traffic that arrives at a node because of broadcasts, misconfiguration or activities that are not directly considered harmful(in that case it would be considered attack traffic), there is a Utwente student paper on this subject from a year or two back which i cant seem to find

[2]SQL injection is an attack that leverages lack of input sanitation to inject SQL sequences into web forms in order to directly manipulate the database behind a web application.

certain attacks low interaction honeypots are often more practical and easier to deploy.

We focus on these properties because we need host-based detectors that are accurate, carry little risk, and can provide a large attack surface for malware to abuse, so that when malware is launching an attack we can be sure it succeeds and that we can capture the details of the attack. Unfortunately this combination does not exist, for example really poor shell code and exploitation skills may fail to exploit a real live system, even if its actually vulnerable, but a low interaction honeypot that merely searches for shell code would still detect the attack. That same low interaction honeypot may fail to detect an indirect attack that attempts to upload a file to the service it only partly emulates, an attack that a high interaction honeypot would have been able to detect, provided it monitored for such events.

2.2.1 Low-interaction honeypot

Low-interaction honeypots typically are system daemons that emulate certain network services and/or network stacks. By doing so, these can capture or detect attacker behaviour, malicious network activity or even capture samples of malware in an efficient way. The low resource utilization means that these approaches are particularly suited for deployment on a large scale where many IP addresses are monitored for malicious probes, attacks or malware.

HoneyD The low-interaction honeypot system HoneyD [40] is capable of emulating operating systems at the network level in order to deceive fingerprinting tools such as Nmap[23], Xprobe2[57] Furthermore it provides the means to create a virtual network on just a single machine, which from a network perspective appears to be a large network containing routers and various operating systems. The HoneyD system is capable of logging all connections and packet signatures of those connections or attach subsystems to ports so a service can be provided. These subsystems can be scripted in python but it is also possible to attach systems such as Nepenthes or high-interaction honeypots as subsystems instead of a scripted service. The main benefits of HoneyD is its ease of deployment, but it is mostly limited to monitoring network activity. The author suggests that it can be used as a front-end for high-interaction honeypots to reduce noise from scanning and other Internet background radiation.

Nepenthes Baecher et al. [4] and Wicherski [74] discuss a low-interaction honeypot called Nepenthes. The purpose of this system is to capture self-propagating malware in an efficient way. It operates by simulating known vulnerabilities on certain ports and listens for connections to these vulnerable services. Malware exploiting these simulated services will try to download a copy of the malware which is captured by Nepenthes. The information

required to download these samples is obtained by feeding shell code captured by a simulated service into a shell code emulator which attempts to extract the download information. Because of this approach Nepenthes is very efficient, but this comes at a price; since it emulates vulnerabilities and lacks a full service implementation new vulnerabilities may not be simulated, and malware utilizing these vulnerabilities will not be captured.

But since the same exploits or vulnerabilities are reused by many different malware implementations even a single module emulating a vulnerable service is able to capture many different samples of malware code. Furthermore because of the way Nepenthes is designed it is possible to, if traffic directed at the honeypot fails to trigger a known vulnerability signature in one of the emulated services, forward the traffic to a live high-interaction honeypot such as Argus for analysis. The results from an analysis can then be used to construct new emulated services for more efficient sample collection.

Network monitors Network telescopes [25] and Darknets[5][60] are an approach to monitoring the Internet for malicious behaviour based on connection metrics. Both approaches monitor a (large) fraction of unused IP addresses and generate statistics about the traffic observed, but do not themselves offer any services other than accepting TCP connection requests. These statistics can be used for instance to track the outbreak of a worm. These approaches ignore the payload of connections altogether, thus they are more suitable as early warning systems. The Labrea tarpit [64] employs this approach to actively combat malware. Instead of just monitoring it keeps connections open and exhausts network resources on the nodes launching attacks in an attempt to slow the spread of malware. An approach with a similar goal is taken in the White Hole [15] system which is designed such that it tries to trick malware into attacking the tarpit.

2.2.2 High interaction honeypot

High interaction honeypots are nodes running a fully functional operating system which optionally runs virtualized to ease management. Since these provide a full operating system(OS) context these approaches allow the exploitation of unknown vulnerabilities. To detect and/or capture these attacks high-interaction honeypots can be monitored in a variety of ways, some approaches of which we will discuss below. However high-interaction honeypots also have their own drawbacks. The use of an operating system, even if it is virtualized, is quite resource intensive and since these are essentially systems like any other network connected node these can also come under the control of an attacker or malware. We will now list and briefly discuss some approaches which utilize high-interaction honeypots to study attacker behaviour and detect malware.

HoneyBow The HoneyBow toolkit [55] is able to capture malware by combining three different sources of information after allowing the honeypot OS to get infected. It operates on the premise that any binary sent to or created on the honeypots filesystem has is likely to be malware. The three approaches are :

- MwWatcher, a (hidden) process running on the honeypot that monitors the filesystem for created files and saves a copy in a specific location on the honeypot filesystem.
- MwFetcher, a process that periodically shuts down the honeypot and checks the filesystem against a reference list for changed or created files.
- MwHunter, a process that carves binaries sent to the honeypot from the network streams. The process of carving data out of datastreams of any kind is usually achieved by searching for patterns in the datastream, typically headers and tails of the filetype being carved. Once found the offsets of these patterns in the datastream are used to extract the file from the datastream.

Furthermore the HoneyBow toolkit provides the means to deploy on physical and virtual machines. This is an important aspect as malware today can employ some forms of virtualization detection and sandbox detection to prevent capture or analysis[6]. The use of honeypots in this way seems similar to the HoneyMonkey[51] system although in that approach other modifications such as changes to registry entries are also monitored and the system is geared towards web-based malware. The major drawback of the HoneyBow approach is that it allows an attacker or malware program full access to the system. Its main feature is that it detects malware that makes modifications to the honeypot filesystem in any way. Since it is relatively simple it can easily be extended to different types of operating systems. One of its limitations is that it periodically shuts the machine down for analysis, which affects the scalability. Another limitation is that it does not provide a correlation between detected malware samples and network traces that caused the malware to be created or uploaded. Lastly it can not monitor for the actual exploit that abused a vulnerable service causing the upload of the malware sample to the honeypot, nor directly tell what service was abused.

Sebek Sebek[69] is best described as an advanced benevolent rootkit developed both for the Linux and Windows platform. The goal of Sebek primarily is to allow extensive monitoring of any activities on a high interactive honeypot. Aside from hiding itself from the operating system it monitors system calls, network traffic and keystrokes to gain a complete picture of actions executed on the Honeypot. An example of a deployment of a high-interaction honeypot using Sebek is outlined in [71].

Argos Portokalidis et al [39] propose a different approach to detecting exploits in a high-interaction honeypot is proposed. Rather than monitoring for file system modifications the actual execution of code is evaluated using taint[29] analysis to detect the execution of shell code. By combining this with recorded network traces it is able to not only detect previously unknown attacks but also generate signatures of such attacks. The authors claim because of the way exploits are detected the system does not generate false positives and can also detect zero day attacks and respond to these quickly by generating signatures for IDS systems. A major advantage of this approach is that it can actually prevent remote code execution on the honeypot. This does not only reduce the risk associated with running a high-interaction honeypot, but also provides better scalability as the OS it runs on does not need to be reset to a previous state. But not all vulnerabilities utilize remote code execution and can therefor go undetected. As such it may be useful to also monitor the OS for signs of infection.

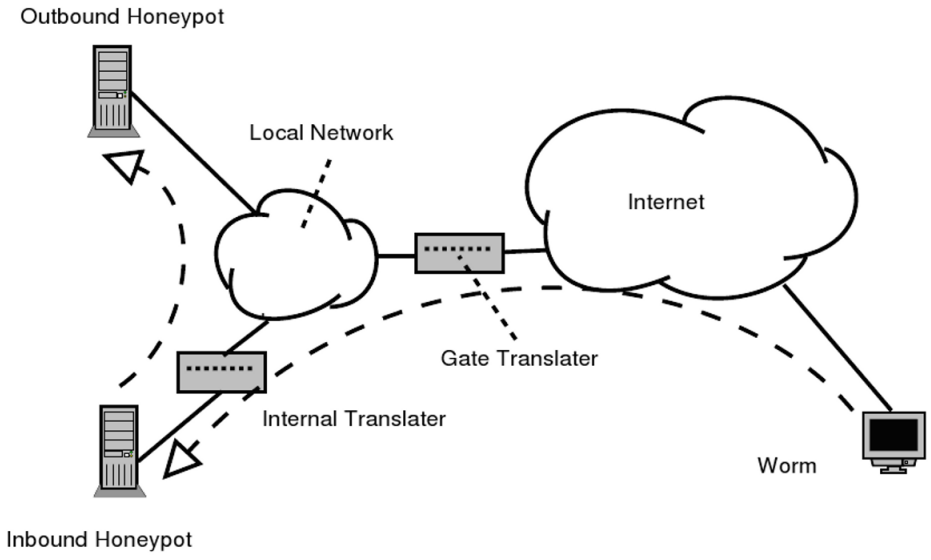

Figure 2.1: Double honeypot system, Translators ensure that attack traffic to unused system/ports is diverted to an inbound honeypot, which in turn has its traffic diverted to an outbound honeypot Image Source: [44]

Double Honeypot The double honeypot system [44], depicted in Figure 2.2.2 proposes to utilize two high-interaction honeypots in tandem in order to detect malware that exploits network services on a local network. To detect

this kind of malware they configure one honeypot such that it is reachable from the Internet and which monitors all unused internal IP addresses and one internal honeypot that is shielded from the Internet. The outward facing honeypot is configured such that by itself it generates no traffic, and all its outbound connections are rerouted to the internal honeypot. This should ensure that any traffic observed at the internal honeypot is generated by malware on the outwards facing honeypot.

Spyproxy concept by Moshchuk et al [27], a clientside honeypot in a virtual machine prefetches URLs in order to determine if pages at these URLs launch any browser based attacks. In essence this is the reverse version of the Shadow Honeypot approach sharing the same advantages and disadvantages.

Other Honeypots in general are hampered by their limited network perspective and high resource usage of high-interaction honeypots. Some approaches to tackle the limitations of the host-based perspective are Honey@Home [3] or Collapsar[17] which utilizes distributed light sensors (distributed network taps) to trap more traffic or HoneyStat[12] which tunes the detection engines such that it can more accurately detect a specific worm on the Internet. To tackle the high resource utilization problem approaches such as Potemkin[47] have been proposed which provides the means to utilize more virtual machines on the same hardware.

2.2.3 Summary

From our survey of honeypots we conclude that while honeypots can be proficient at detecting malware that propagates by exploiting or abusing network services there is always a trade off between scalability and detection rates.

In general we observe the following limitations of host-based malware detection techniques :

- Honeypots can be located and therefore avoided, and can only present a view of the traffic they can observe. This makes a honeypot-based approach to protecting or monitoring a local network particularly unsuitable because only malware that actively scans the local subnet is likely to launch an attack on the honeypot. Injecting honeypots at random spots in the IP range of a network can give a higher chance of observing an attack, but the problem is not fixable by improving the chance of selecting a honeypot when a target is randomly selected. The assumption with such an approach is that any malware attempting to exploit other nodes on the network will scan or attack the network by randomly selecting targets or by simply scanning the entire IP space in a linear fashion. Other propagation models have been proposed[31] that do not exhibit this behaviour, for instance by using hit-lists, fingerprinting for

honeypots or by relying on the ARP cache which allows an attacker to avoid honeypots.

- In the case of high-interaction honeypots patch levels and the operating system used govern if certain vulnerabilities are present. This in turn could result in malware not being detected as such because a vulnerability is not present.

- While some honeypot approaches are advanced, none of the approaches are able to detect every type of malware.

- In order to be able to detect previously unknown attacks high-interaction honeypots are essential, but high resource utilization makes it impractical to use these on a large scale, and also makes it harder to match a suspected attack with the right honeypot, as the range of attacks a high interaction honeypot is vulnerable to can never be as broad as an emulated honeypot. For example its perfectly viable to implement an emulated web service capable of detecting known attacks on both the Apache and Microsoft IIS web servers.

Unfortunately none of the surveyed approaches will detect all possible forms of an attack simply because an attack may fail altogether because the honeypot was not vulnerable to begin with or because it failed to detect an attack took place. In table 2.2.3 we summarised the various approaches. Its not sufficient to just use the low interaction emulation based approaches, even though these are capable of emulating every known vulnerability its the unknowns that give rise to the most worry. Similarly the high interaction honeypots can fail to detect an attack has taken place, Honeybow may fail because the attack avoids touching the honeypots filesystem,or Argus may fail because the attack does not use a memory corruption attack.

This means that to increase the chances of detecting an attack its not sufficient to rely on just one honeypot approach, low interaction honeypots such as Nepenthes can deal with a large number of attacks, but high interaction honeypots monitored through a number of the approaches listed in 2.2.3 built upon several different Operating systems seems the only way to assure a large coverage.

Honeypot	High/low interaction	Automation	Detection method	Scalability	Risk	Remarks
HoneyD	Low	High	Emulation	High	Low	
Nepenthes	low	High	Emulation	High	Low	
HoneyBow	High	High	Filesystem monitor	low	High	May not detect all malware, allows execution of malicious code
Sebek	High	Low	Manual	Very Low	Very high	More suited towards investigating manual attacks
Argus	High	High	Taint analysis(of injected code)	Medium	Medium	Prevents malicious code execution
Double Honeypot	High	High	Traffic analysis	Low	High	Can detect any malicious traffic, can be safe if properly contained.
Network Monitors	Low	High	Traffic analysis	High	Low	

Table 2.1: Comparison of malware detection and capture systems

2.3 Network-based detection

A network-based approach monitors any traffic being routed through a particular network link which allows for an integral approach to analysing network traffic, as opposed to the limited perspective offered by host-based approaches. By applying signatures or statistics these approaches try to separate network traffic in malicious or non-malicious streams or monitor for specific events such as attacks on network services. Network-based approaches are generally referred to as (Network) Intrusion Detection Systems or (N)IDS for short.

Analysing network traffic is difficult because of the high volume of data that must processed. An IDS can either be deployed in-line where it analyses packets and makes a decision whether to forward it or be deployed in logging mode, where the IDS does not interfere with the routing of packets but merely logs them for inspection. In either case processing time is critical as to prevent the network service from degrading. In in-line mode the IDS must process and decide what to do with a packet within a certain time frame to ensure proper network operations. In packet sniffer mode these constraints can be relaxed somewhat, but the IDS must still be able to process traffic within a reasonable time frame as generally problems or attacks that arise must be detected and dealt with as soon as possible. So in order to design an IDS that can cope with a large volume of traffic it is therefore imperative that operations used can either be distributed easily and/or are computationally inexpensive.

In the literature two main approaches to analysing network traffic can be observed which either employ signature matching which monitor traffic for specific fixed features and anomaly-based approaches which build statistical models of certain features of network traffic in order to detect malicious traffic. Since we are specifically interested in detecting malware that spreads by exploiting vulnerable services, which is a subset of the malicious traffic these approaches can detect, we will briefly discuss these approaches in the remainder of this section.

2.3.1 Signature-based approaches

Signature-based approaches attempt to detect malicious traffic by monitoring traffic for predefined (fixed) patterns of malicious traffic, these patterns can consist of connection statistics, packet payload or specific protocol interactions. The two most well known signature-based systems are discussed below.

Snort Snort [41] is a packet-oriented signature-based IDS, its architecture is depicted in Figure 2.2. The Decoder is responsible for decoding network layer protocols and placing these and the packet payload in suitable data structures for easier processing. The pre-processors can be inserted as needed and can perform various network oriented functions such as fragmentation reassembly, anomaly detection on traffic metrics(SPADE) or TCP stream reassembly.

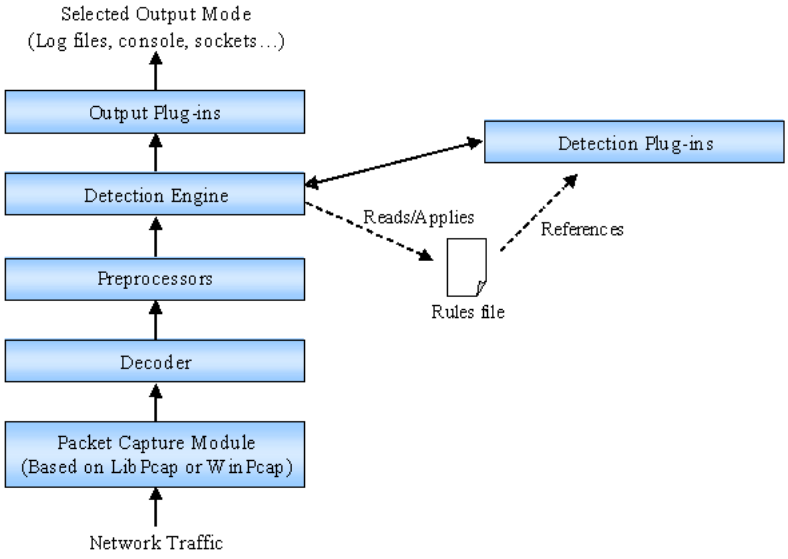

Figure 2.2: Snort architecture (Source csdn.net)

The main component of Snort is the Detection Engine which utilizes a rule-set to detect malicious signatures by matching the patterns defined in the rules with traffic received from the pre-processors. The initial version was stateless and could only perform signature matching, however the current versions of Snort are also capable of maintaining protocol state which for instance allows the detection of attacks that span multiple protocol interactions. To facilitate the integration with existing IT infrastructure snort provides plugin-based output mechanisms. Its modular design makes it also a useful framework for research since it can easily be extended by adding pre-processors, Detection or Output plugins.

Bro [32] is a higher level signature-based IDS system. It is high level in the sense that it is capable of tracking and interpreting events at the application protocol level. The architecture of Bro is depicted in Figure 2.3.1 and is divided into four components. It utilizes Libpcap to capture packets and filter out those that should not be forwarded to the Event engine based on destination ports. It's main purpose is to filter out all packets containing a higher level protocol

that Bro does not know how to interpret. Aside from filtering for specific protocol ports it let's through packets which have either the SYN,ACK,RST flags set or packets that are fragmented, this in order to reduce the load on the event engine while still capturing entire protocol interactions and important network events.

The Event engine is responsible for transforming packets and/or streams into higher protocol level events, for example HTTP GET requests observed in a packet would be classified as an event. The policy script interpreter is responsible for tracking events and matching these to the rule-set that defines the characteristics to look for such as specific patterns. The dotted reverse arrows are used to control the granularity of the filters of the lower level component to achieve higher throughput. For example there is no point in having the event engine generate if no rule for such an event exists in the policy.

Similarly, protocols that are not analyzed by the event engine do not need to be forwarded by the packet filter and can be dropped there. The caveat from the packet filter and this approach in general is that non standard protocols or traffic on non standard ports will not be analysed unless these are explicitly added to the policy or the event engine.

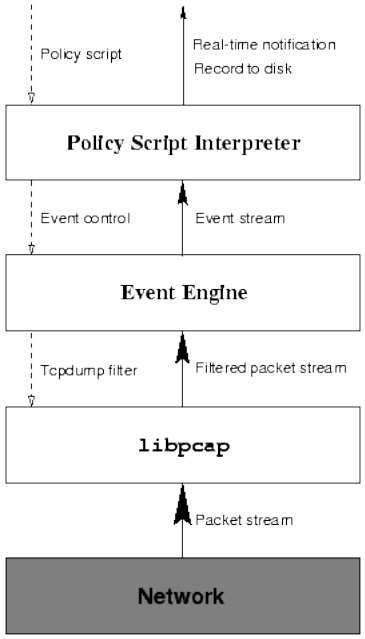

Figure 2.3: Bro high level architecture source :[32]

Signature generation Bro & Snort rely heavily on predefined rule-sets which describe which patterns indicate malicious traffic. In order to reduce the amount of manual labour involved in writing these rules and reduce the reaction time to new threats a number of approaches to automatically generating signatures for these types of systems have been proposed. We will summarize a few well known approaches below and discuss why these are of limited use in protecting a local network.

Host-based signature generation **HoneyComb** [20] utilizes Honeyd to trap incoming malicious traffic and to generate signatures based on multiple longest common strings found in the traffic. **Sweetbait** [38] utilizes a more

sophisticated honeypot, Argos, to trap incoming malicious traffic. By simultaneously monitoring the network traffic to the honeypot the system is able to generate signatures based on string similarities or protocol features of the observed traffic.

Network-based signature generation All the network-based approaches to signature generation described below rely on an external detector to classify TCP streams into a pool of malignant and non-malignant streams. From the pool of malignant streams these approaches attempt to extract signatures based on feature similarities between streams. This approach is founded on the notion that attack traffic, particular attacks on network services, contain some invariant features which are always present when the attack is executed. The PAYL[48] anomaly detection system generates signatures from the pool of malignant traffic, it is designed such that it can detect similarities between packets going to and from a host. If such a similarity exist there is a high probability that this are caused by similar malware traffic, such as worm activity and the system attempts to generate a signature for these packets based on string similarities.

Autograph [1] also generates signatures by analysing malignant flows for common substrings. A difference with other approaches is that it dynamically computes how the TCP stream should be divided into variable length *content blocks*. It will attempt to match these content blocks with other suspicious TCP streams, if the number of matches of content blocks is above a certain threshold these are used to generate a signature. Because Autograph computes the size of these content blocks dynamically these may be to short or overly long , which is generally undesirable, and therefore also provides upper and lower boundaries for the size of these content blocks. The approach was criticized for having limited capabilities for dealing with polymorphic worms.

Polygraph [30] is a proposal to generate more accurate signatures for polymorphic worms. Rather than just matching byte sequences this approach creates signatures that describe a polymorphic worm in an ordered or non-ordered sequence of separate byte patterns. This idea is founded on the notion that every worm or exploit has to have some characteristic features even polymorphism can not eliminate, and which are heavily defined by the type of exploit used. They combine this approach by proposing a scoring system of certain byte patterns that can match a sample of a polymorphic worm. If the scoring is high enough the signature is considered to match the sample under investigation.

Hamsa [22] Is an approach that also extracts features from malignant streams to form signatures, but claims to be more efficient at doing so than Polygraph. It also seems to be able to deal with more noise in the data to analyse.

Summary There are two main issues with signature-based approaches. First and foremost these systems require an extensive set of signatures of malicious traffic. Creating good signatures that only match malicious traffic in itself is already difficult, but the problem is compounded by the fact that there is a limit to the number of signatures can be processed. Already approaches like snort or bro utilize a strategy which attempts to avoid applying certain groups of signatures if these are not relevant.

Furthermore it has been shown [28],[46],[33] that there are limits to automatic signature generation and that signature-based detection can be evaded. For example, in order to detect polymorphic worms even the more advanced approaches still rely heavily on specific features of a worm. But it has already been demonstrated that it is possible to construct worms which do not have these fixed features, although this only seems to apply to exploits that avoid abusing a memory corruption vulnerability[45]. Also these approaches are pattern-based, which means that if there exists a malicious stream with certain features, and another non-malicious stream with the same features the signature-based IDS cannot make a distinction between the two.

To summarize, the use of signature-based IDS systems augmented with automated signature generation does provide a high degree of detection for some classes of malicious traffic, however these approaches still seem to struggle with achieving high detection rates while maintaining a low false positive rate and are particularly challenged by attacks which employ exploits and encoding techniques which can avoid the use of fixed features.

2.3.2 Anomaly-based approaches

Another main area of research into detection of malicious network activity is the use of anomaly-based detectors. These detectors build profiles of what constitutes normal traffic and raise alerts if a network flow is significantly different from the standard network profile. For example in HTTP requests there exists a specific distribution of ASCII characters, should a program attempt to exploit a web server this is often followed by injecting some shell code which alters this distribution significantly and can thus be detected. Another example along the same line is that following an HTTP GET request there will be a response from the server with a page of a certain size. A server that is exploited most likely will not return such a page because its program flow is altered. The absence of the response from the server constitutes an anomaly and can thus be detected.

Like signature-based approaches, anomaly-based detectors need to be configured prior to deployment. In this case the detectors build models of the traffic they will be monitoring, and later will compare such models to incoming traffic in order to detect anomalies. Clear advantages of anomaly-based approaches

are that these are self-learning and can detect new unknown attacks as soon as they take place.

Anomaly detection systems can be divided into three general approaches, systems described in literature generally employ one or more of these simultaneously:

- Anomaly detection based on packets headers[24][13]. These focus on specific features in packets such as flags or build profiles of the number of connection requests or destination distributions.

- Content(payload)-based approaches[49][50]. An approach where profiles are constructed of the content of streams or packets, for example the previously mentioned character distribution in an HTTP request is an example of this.

- Correlation-based approaches.[7]. An approach which correlates inputs from different NIDS approaches in order to provide more accurate alerts. The principle idea here is that single alerts raised by any kind of IDS by themselves lack context, for example one failed TCP/IP connection does not constitute a scan in most cases. But correlating such events may make a stronger claim that these events are part of a port scan. The approach taken in [7] combines an existing IDS monitoring incoming traffic with a correlation system and anomaly detecting observing outgoing traffic. By correlating alerts raised on incoming traffic with changes observed in outgoing traffic it can more accurately remove false positives raised by the inbound traffic NIDS monitor. These correlation based approaches are also employed specifically for detecting botnets, this because botnets typically will exhibit several measurable features that can be observed using NIDS systems such as scanning, exploitation and command & control traffic. By correlating these events botnet detectors can make more accurate detections then when treating each event separately. Some approaches that employ this technique are discussed in the next section.

Anomaly-based detection shows promise and is actively researched. For now payload or packet header based approaches are the most common because these are versatile, but the same approach could also be used for detecting anomalies at the application protocol(semantic) level. Since current approaches do not interpret data at the semantic level it is not possible to assess directly if a particular packet is indeed malicious. Instead these approaches rely on the notion that an anomaly is caused by malicious activity as malicious activity generally constitutes a divergence from normal behaviour. This is important for our approach as an anomaly detector in principle can not distinguish between an anomaly generated by a malformed GET request to a web server or anomalies generated by a node attempting to exploit a certain vulnerable service.

This leaves the anomaly-based approach open to malicious traffic that mimics normal traffic behaviour [19][14] or attacks on the detector where an attempt is made to alter the model it uses to classify data in such a way that attack packets are not detected as an anomaly. A second issue with anomaly-based detectors is the learning or training phase. In this phase the detector is fed volumes of traffic which it uses to construct a model. Depending on the detector the traffic used to train the model may contain a certain amount of malicious traffic, called noise. It has been demonstrated [11], [10] that the accuracy can be improved if training data is used with little or no noise. Constructing a training data set free from attack traffic from real data however is labour intensive.

Finally a difference between some approaches is the time it takes to classify traffic as malicious, some approaches employ a scoring system or analyse entire TCP streams rather then a packet or group of packets and or couple this with a scoring system in order to make an accurate assessment. This improves accuracy but also incurs a delay over approaches that classify individual packets directly as they are received.

2.3.3 Botnet detectors

The distinguishing feature of botnets compared to families of malware is that botnet malware must at least partly comprise of semi-autonomous malware which is controlled via a (multicast) unified control structure. i.e. it is malware programmed to automate the execution of certain malicious tasks but only after after the malware program has been instructed to do so and been provided with the relevant information it needs to execute a task. This entails that there always exists a specific sequence of observable (network) events when botnet malware is considered. For example before a malware bot can begin sending spam it must first receive the instructions to do so via a communication channel, then proceed to fetch the spam data and then proceed to spam the targets on the list it received. Because botnets are common and pervasive and are the basic building block for criminal activity on-line it pays to focus on botnets and the specific features exhibited by botnets.

Approaches to botnet detection are a specific application of correlation-based detection approaches leveraging multiple network-based analysis approaches for information. These correlate events generated by signature-based and anomaly-based detectors in order to detect malicious activity in network traffic. Botnet detectors in particular search for particular sequences or combinations of events indicating the presence of botnet malware.

For example malware randomly attacking other nodes often exhibits the sequence of scanning for a vulnerable service and/or exploitation of this service followed by subsequent downloading of malware to an exploited node. By combining these events alerts can be generated more accurately with more confidence and with more confidence. Some events also make a stronger case

for the presence of malicious traffic or malware, for example a failed TCP connection is not as strong an indicator as an attack on a vulnerable service on a honeypot. So correlating multiple (weighted) events allows assigning of importance to individual events that may sum up to a real indication of malicious activity.

The field of detecting botnets is also a special case because botnets typically utilize a Command & Control structure. This is significant as it not only introduces features to search for in traffic, but because it introduces traffic not normally generated by non-malware programs. This adds an extra feature to add to the model which describes malicious traffic generated by botnets. We will discuss two approaches that employ these features to detect botnets below.

Bothunter[16] The BotHunter system specifically targets botnet malware through correlation of IDS events. It achieves this by modeling the activities of botnet malware as a specific sequence of events to which it assigns a certain weight. These events are generated using traditional signature and anomaly based approaches that search for command and control traffic, scanning traffic or traffic that contains signatures of exploits or executable binaries. By observing some of these events in the right order and whose summed weight is higher than an a predefined threshold it is able to discern botnet traffic from normal traffic.

This approach allows the use of indicators of malware activity which when used by themselves are not accurate enough. For example flagging every TCP stream that contains an executable windows binary is not a useful approach in itself for detecting bot traffic, as the occurence of such an event does not provide strong evidence of a malware infection. If however this event was preceded by an scan of this machine and subsequently this machine exhibits scanning behaviour itself then a stronger case can be made for assuming the binary(and the scanning behaviour) was malicious.

Automatically Generating Models for Botnet Detection Another recent [53] correlation based approach focuses on detecting and correlating the initial command that causes bots to launch attacks with an observed change in network behaviour. The approach followed is that the network flows are monitored for changes that indicate scanning behaviour such as changes in volume or connection count. If such a change is detected the system then tries to correlate this with command and control commands sent at an earlier point in time.

The signatures and models used to search for the commands sent to and responses of the botnet are constructed by profiling known botnets in a controlled environment. Wurzinger [53] demonstrates that this approach is effective at detecting botnets in laboratory settings and in this environment outperforms

the approach followed in the Bothunter system. However its detection capability relies heavily on being able to extract accurate features from command and control flows, which more or less requires an unencrypted channel, and requires training with captured botnet malware before deployment.

Evaluation Even though botnets are a specific subset of malware these approaches provide valuable insights towards approaches which detect malicious network activity in general. A survey[43] of several approaches to botnet detection discusses how these approaches can be circumvented. This survey argues that the approaches to combating botnets should focus on features that are inherent to botnets, rather then focusing on specific traits exhibited by certain malware families. As it is focusing on universal traits, like synchronized execution, are to be preferred over targeting string similarities in command structures or similar target systems which. This follows from the relative ease with which the later approach can be defeated, but changing the observable behaviour is far more difficult and not easily hidden via obfuscation or encryption. In essence the focus should be on (observable)traits or behaviour that are inherent to the way malware functions and achieves its goals.

A serious problem that emerges from our short survey of botnet detectors is that there seems to be a lack of good testing data with which to compare(benchmark if you will) the accuracy of the detections. Nor is it clear what is causing the detections approaches to detect some malware samples better then others or why samples are not detected at all. We suppose that current techniques suffer still because of using relative high thresholds for analyzing network traffic to avoid false positives and rely (to heavily) on signatures for generating events. I.e. it seems the detector would discard evidence of malware in favour of improving the false positive rate of a detector.

2.3.4 Other

Our main area of interest is accurately detecting hosts that spread malware, in that light we will discuss some approaches that do not really fit into any of the previously discussed categories for network intrusion detection but are relevant for our research.

The Shadow Honeypot concept by Anagnostakis et al [2]. In this approach anomaly-based network detectors are placed before a (web)service in order to detect attacks launched at this service. Any traffic that triggers the detector is directly diverted to a high interaction honeypot which provides a duplicate of the (web) service and and the system providing the service. This high interaction honeypot is fitted with various detectors or instrumentation such as Argos and service specific detectors enabling it to detect a variety of attacks that target the protected application.

This is an attractive approach as any false positives generated by the anomaly detector, or failed conversions of true positives by the honeypot do not affect the system. In case of a false positive the service is provided at a slower pace, but it is still available to the user. In case of failed conversion either an attack failed to execute or be detected by the honeypot. Since the context of the honeypot and the real system is the same an attack would presumably function on both contexts or none of the contexts. Detection is limited to attacks which target the this system, and which make use of vulnerabilities in the context of the service or operating system providing the service. The idea we explore shares the same concept, but we aim to detect (any) attacks close to the source where these attacks originate from. Not every honeypot is capable of detecting any attack, so this entails using several (different implementations of) detectors and formulating some strategy for redirecting attacks to the proper honeypot detector.

This approach is different from our approach because of different design goals and resulting different architecture. In the hybrid detector we must predict the nature of an attack and what type of context should be made available for this attack to succeed and become detectable whereas the shadow honeypot implicitly assumes the service/system providing the service is the only context for an attack. This is not an issue for the approach as the focus is on detecting (only) the attacks that affect the service, rather then localizing malicious sources by detecting every attack.

Bait and switch A snort module that is no longer maintained implements a scheme to switch traffic to a honeypot when firewall rules are triggered. There exist two implementations with the same name, an original implementation [73] and a second (re)implementation [66]. At its core our approach is similar, though we extend the approach to anomaly based approaches and devised a scheme which decides to select detectors to which traffic should be routed to.

Network level emulation Polychronakis et al.[35],[34] and [36] present an approach that attempts to detect exploits in packet or TCP streams by searching for signs of executable (shell) code using an emulator. It is different from existing approaches because it deploys techniques to identify code in streams which will actually execute without interacting with the network stream under analysis. It couples this machine code identification with a basic heuristic that looks for the usage of certain instructions to classify code as a malicious payload. Using this type of detection false positives can effectively be eliminated, but not all types of malicious code may execute within the emulator because it is missing the context of a real operating system or may not be detected as malware because it does not employ the instructions the detector monitors. Even though it may not be able to detect all types of malware this approach has a significant advantage over traditional honeypots as it does not have to provide a vulnerable service or a specific OS and patch level. Some

approaches [58] aim to hamper this type of detection by trying to avoid triggering the heuristic rules used by the detector, try to break the code emulation by relying on unsimulated functionality or using context only available on the targeted machine.

A similar approach [54] combines static analysis and emulation to detect the presence of an encryption routine in binary data. The reason this particular feature is targeted is that it is generally only employed by programs that attempt to evade IDS or prevent analysis. A reasonable assumption here is that if these programs need to hide from these kind of systems they are of a malicious nature.

Sword Li etal [21] present an approach to detecting Internet worms is discussed which tries to match network traffic patterns against predefined heuristics of known worm behaviour. It flags traffic as worm traffic if there exist a causal similarity between connections, for example a connection is observed to a particular node and subsequently outbound connections with similar signatures originate from that same node. It also analyses the traffic destinations from individual nodes which lets it detect worm traffic based on differences observed in the distribution of traffic destinations.

There appears to be little work done on comparing the detection capabilities of network-based detectors and honeypot implementations. In their master thesis Keemink and Kleij [18] survey the detection capability of Snort and honeypot based on experiments using live traffic. The SurfIDS system has network traps similar to Collapsar[17] that gather traffic and redirect these to the Surfnet system. The experiment consisted of analysing this captured traffic by diverting it to honeypots while simultaneously analysing the traffic using the Snort detector. Their results would suggest that, for their experiments, the detection capability of the network detector and host-based detectors only partly overlap.

2.4 Network statistics and containment strategies

A detailed evaluation of post infection network activity can be found in [37], which surveys connections made by malware after a machine was infected. This research is interesting because it identifies certain traits that could indicate that a node is infected with malware. One observation they made is malware often sends out email from the infected host or uses some other approach like HTTP to report information back to an attacker. They observed that this is sent directly to an SMTP host that is not the relay for the current domain, which means this feature could be used to detect an infected host. Another

characteristic feature to monitor is proposed in [52] is to monitor the amount of TCP SYN packets a node sends out. A high count could indicate malware activity, but this approach will not detect worms that intentionally keep their connection rates low.

Containing self propagating code [26] makes a case for containment of self spreading malware on the Internet over patching or treating the symptoms.They ask themselves the question: is it even possible to contain a worm and how effective can this be? They conclude a system has to be able to react automatically and employ content filtering rather then source-based filtering. Containment strategies for worms are evaluated by Brumley et al. [8] An interesting conclusion is that local containment of worms via throttling connections, even when deployed on a large scale, is not sufficient to slow down a computer worm epidemic by more than a factor of 10. Using mathematical models it is further demonstrated that fast inoculation against an infection via patching or virus signatures and hardening systems to make infection more difficult will slow down the spread of a worm more then blacklisting sources infected with worms or local containment strategies via connection throttling. Their model assumes that new insights into hardening operating systems and software and the distribution of known malware signatures and patches happens uniformly and timely, something which does not seem possible on today's Internet.

2.5 Conclusion

Traditional host-based malware mitigation approaches such as providing adequate patch management or utilizing various anti-virus software have proven insufficient. This not just because there currently always exists malware for which no signature is yet created but also because the time window in which these need to be applied or updated is ever decreasing. Host-based anti-exploitation techniques such as hardening the OS can mitigate some of the threats these too can not yet guarantee complete resistance against malware infection. So we surveyed approaches that focus on monitoring network traffic for signs of malicious traffic generated by malware and observed the following:

- Host-based detection techniques are efficient at detecting malware that launches direct or indirect attacks on their services but are not sufficient to combat malware because of a limited network perspective, furthermore to achieve detection the honeypot must present the right OS and service for an exploit to function. Lastly a honeypot can only detect but not stop malware directly, though it may be able to keep systems launching attacks occupied with attacking the honeypot instead of real targets.

- Signature-based intrusion detection systems can detect malware in network streams and additionally classify what was detected, but can be

evaded by employing polymorphism techniques and/or attack vectors which lack the distinct features these approaches require to ensure a proper detection. Some mitigation's exist but rules must be carefully selected to not match normal behaviour or non-malicious network streams. It

- Anomaly-based approaches promise better detection of previously unidentified threats with low false positives resulting in a faster reaction time to new threats than signature-based detection approaches. A caveat of these techniques is the limited ability to classify the nature of the anomalous behaviour. We are only interested in a specific subset of anomalous behaviour, namely the behaviour that is actually malicious, whereas anomaly-based approaches will flag anything that is out of the ordinary.

- Network detection approaches that do not actually verify the malicious nature of observed traffic are themselves vulnerable to attacks which attempt to generate large amounts of false positives in an attempt to masquerade a genuine attack.

- Network level emulation techniques can detect a particular type of shell codes rather well, but are resource intensive and only applicable to certain shell codes and malware distribution scenarios.

- We see that correlation approaches that combine inputs, for example like employed by Bothunter, can provide a substantial improvement in the accuracy of detections made, but for the purpose of stopping malware attacking other systems these have some limitations. First of all correlating and compounding events takes time which makes such an approach suitable for detection, but not for preventing attacks. Also the approaches we surveyed are still based on traditional anomaly/signature-based based NIDS forcing the use of thresholds and specially tuned signatures, this to minimize false positives, resulting in a loss of detection capability.

Property	**Anomaly**	**Signature**	**Honeypot**	Description
Monitor scope	Network	Network	Host	Range of network traffic monitored
Ability to detect unknown attacks	Yes	No	Yes	Only high interaction honeypots can detect unknown attacks
Requires apriori attack details	No	Yes	No	
Requires training data	Yes	No	No	
Alerts provide context/attack details	No	Yes	Yes	Anomaly detection can not indicate why traffic is anomalous
Requires manual review of alerts	Yes	Yes	No	reviewing of events causing false positives

Table 2.2: Feature comparison of anomaly and signature-based network detectors and honeypot detectors

No approach will detect every instance of malicious traffic or be immune to evasion, even if a detector specializes in a specific type of malicious traffic. Its also seems that signature based approaches are not sufficient to deal with unknown types of malware (traffic) and in many cases can be evaded. Anomaly based approaches seem to be the most promising in this respect as evasion or blending attacks are difficult and the dynamic nature allows detecting of unknown threats. The detection rates of anomaly detectors are coupled to the error rate, which is what presents some issues with these approaches. If deviation from standard behaviour is defined very restrictively, i.e. the smallest deviation is labeled to be malicious its likely to detect many instances of malicious traffic, but make many errors doing so. Coupled with the characteristic that anomalies in principle are not typed in any way this makes further refining of detection events from an anomaly detector difficult, as it flags only on anomalous events, not malicious events. This is why it seems anomaly detectors can benefit greatly from end point detectors interacting with potential malicious streams that are capable of confirming and classifying the malicious traffic. Examples of end point detectors would not just be traditional honeypot approaches but also web-application-firewalls or filtering proxies. These endpoint detectors typically classify based on rules applied to observed changes which occur as a result of the detector interacting with the malicious stream, as opposed to inspecting the content of the stream directly.

This makes a combination of anomaly detectors driving traffic to host-based detectors an interesting concept to explore as it allows the extension of the scope of host-based detectors or affords strengthening anomaly detectors by coupling anomalous events with context information from host-based detectors.

Chapter 3

Hybrid architecture

Why do hybrid detection The advantage of using a hybrid approach is that it extends the capabilities and accuracy of both the network detectors and honeypots. This is achieved via traffic redirection. In order to perform traffic redirection some control over the network link layer is required, we discuss this separately in Section 3.3.4. For the remainder of this thesis we assume that full control over this layer exists and we have full control over traffic flowing in and out of each individual node connected to the local network.

Extending scope of honeypots By redirecting traffic into honeypot detectors these can actively hunt for malware traffic instead of passively waiting to be contacted. It also presents malware with weak targets which should encourage attacks on the detector and at the same time quarantines the malware because it is no longer attacking other targets. We can improve the chances of observing an attack by utilizing different honeypot implementations as detectors.

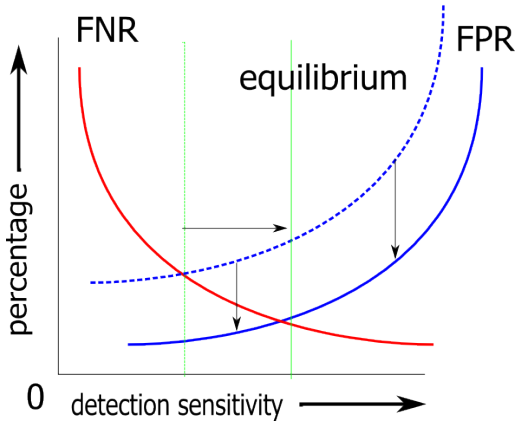

Figure 3.1: Example graph of an (idealized) detector showing the relation between false positive rates (FPR) and false negative rates(FNR), which are defined as a percentage of errors made when classifying data.

Improve detection rates It is impossible for the second stage detector to directly improve the detection rate of the first stage detector since it can only analyse traffic that is already flagged by the first stage. Network-based detectors however make trade-offs between the detection rate and

the false positive/negative rates. It is here where improvements can be made. Since a detection in the hybrid approach by the network detector is not final until confirmed by a second stage detector we can eliminate some of the "uncertainty" regarding the accuracy of a detection. This entails that from the set of alerts raised by a network-based detector we can filter out false positives if alerts did not lead to a detection by the second stage. This may allow a detector to be tuned to a higher sensitivity level which normally would lead to an unacceptable false positive rate. This concept is illustrated in Figure 3.1, the equilibrium is the sensitivity level where the false positive/negative rates are still acceptable. If we can detect and correct a false positive by the first stage detector this means the false positive curve shifts downward, affording a higher sensitivity level and a higher sensitivity level can be used. This is not a completely free lunch, as the false negative rate may shift upwards if second stage detectors fail to confirm true positive alerts from the first stage detector. The amount of confirmed true positives generated by a first stage detector we call the conversion ratio. By conversion losses we mean the True detections by the first stage that are not confirmed by the second stage. Ensuring a detection if malicious traffic is present is discussed in section 3.3 on detection strategies.

Detect unknown threats Both anomaly detectors and high interaction honeypot detection systems are capable of detecting new threats, and combining these would overcome the systems individual deficiencies, lack of accuracy of anomaly detectors and lack of scope for honeypot based detectors.

Limitations of a Hybrid detector(why not do hybrid detection) Hybrid detection has the following limitations:

- Quality of Service is compromised, the system actively reroutes traffic to different destinations thereby (temporarily) disrupting communication flows. This is something that can only be partially mitigated by optimizing detectors as anomaly based detection approaches can not avoid making errors.

- Discerning an unconfirmed True positive from False positive. This means that a malicious event that was not detected by the second stage, IE a true positive which is not confirmed, is not distinguishable from a false positive from the first stage detector.

- Limited ability to detect malicious traffic that is not part of automated attacks. An attacker could confuse or avoid the detector by sending enough attack traffic to trigger the 1st stage detector and then stop until it is no longer redirected to the 2nd stage detector. This is partly mitigated by the fact that an attacker can not predict how long or when its attack traffic is being intercepted and redirected.

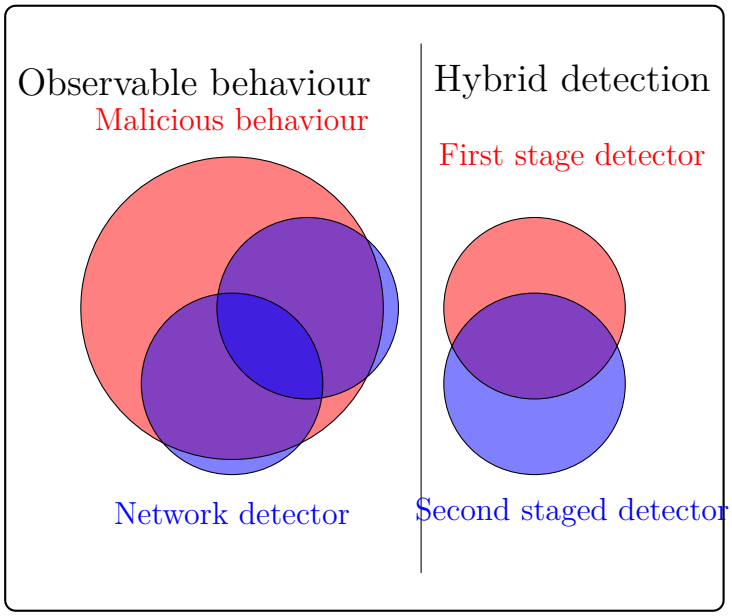

Figure 3.2: Venn diagrams demonstrating the relation between malicious behaviour and detectors. Left side showing the relation between malicious behaviour and network detectors, on the right the relation between first and second stage detectors.

- Only malware that launches enough scans, attacks or performs malicious actions to trigger an anomaly detector and a detection by a honeypot can be detected by this approach. The number of attacks required for detection is at least the number of attacks required to trigger an anomaly detector and a connection containing an attack to a second stage detector.

3.1 Design Goals

The goal of this architecture is combining anomaly based detection with honeypots such that:

- The accuracy of anomaly based system can be improved. This is achieved by providing a way to assess decisions made by, and/or make corrections to the model used by, the anomaly detector. This may also alleviate the trouble with training anomaly detectors.
- The scope of honeypots is extended thereby improving the odds of honeypots getting attacked by malware and obtaining samples of the attacks.
- It is easy to utilize or evaluate different anomaly detectors and honeypot systems as first and second stage detectors.

- Malware relying on the network to spread can be contained without requiring prior knowledge of that malware. It achieves detection in a generic way by using network analysis to search for patterns or anomalies malware traffic will exhibit and redirecting traffic to detectors that must be vulnerable. This makes it applicable to a wider range of malware without requiring reconfiguration and makes it difficult to circumvent.

There are also some usability issues to consider:

- Maintain network connectivity, maintaining quality of service is important. Since we are actively rerouting network traffic to a different destination this will disrupt normal communications. Disrupting communication flows is unavoidable with second stage detectors such as honeypots so instead we must rely on a strategy that aims to minimize the impact of a false positive generated by the first stage detector. If the system makes many errors and takes a long time to correct these it becomes an impractical nuisance instead of a useful defence mechanism.

- Automation, in order for systems to be useful they need to be able to operate with a high degree of autonomy avoiding the need for constant input from administrators or continuous updating. Ideally a system should just indicate what nodes are infected, isolate these keep these quarantined until the infection is removed.

- Simple to deploy, the key to limiting the spread of malware is by deploying monitoring systems that limit the ability of malware to attack other systems. The only way to create networks that hamper malware is by simplifying the management and installation of detection systems, as high complexity will hamper deployment.

3.2 Functional description

In this section we describe the hybrid architecture which implements the decisional structure depicted in Figure3.3. We break the system down into individual components, depicted in Figure 3.4, based on how these components manipulate network traffic and their responsibilities. In the image we included control lines to indicate which different components interact and red arrows indicating the flow and volume of traffic. For each component we enumerate the input and outputs of message exchanged between components and ingress/egress network traffic. In the remainder of the section we summarize how we expect to be able to capture enough traffic for analysis and how we expect to control traffic flow to and from individual nodes.

3.2.1 Network traffic capture

This component is responsible for aggregating network traffic so that it can be analysed by network detectors or redirected to honeypots. In practice this requires some changes to the network infrastructure or the way traffic is routed. This can be achieved using network management functions or MAC spoofing attacks (see Section 3.3.4). Treating the network as a black box from which we can tap traffic and have traffic intercepted just serves as a practical abstraction. It serves the following purposes:

- Represent a source of network traffic for network detectors to analyse. The make up of this network traffic should be such that both internal and external communications of local nodes can be monitored. The goal is to provide the network detector with as large a portion of network flows to each individual node as possible.
- Provide link layer traffic rerouting to aid in investigating and quarantining suspected nodes. This redirecting is vital because without it is not always possible to rewrite traffic such that it can be routed into honeypot detectors. The effect of this rerouting is that it gives the network redirecting component control over traffic as if it were placed between the node under investigation and the network.

It may not be possible or practical to set up a system where all traffic is analysed or all traffic to and from any node can be controlled at will. This does not change how the hybrid system works, but it does mean it takes longer and becomes harder to detect malware as network detectors have less traffic to analyse and potentially fewer malicious connections can be intercepted and redirected to second stage detectors.

Inputs/Outputs

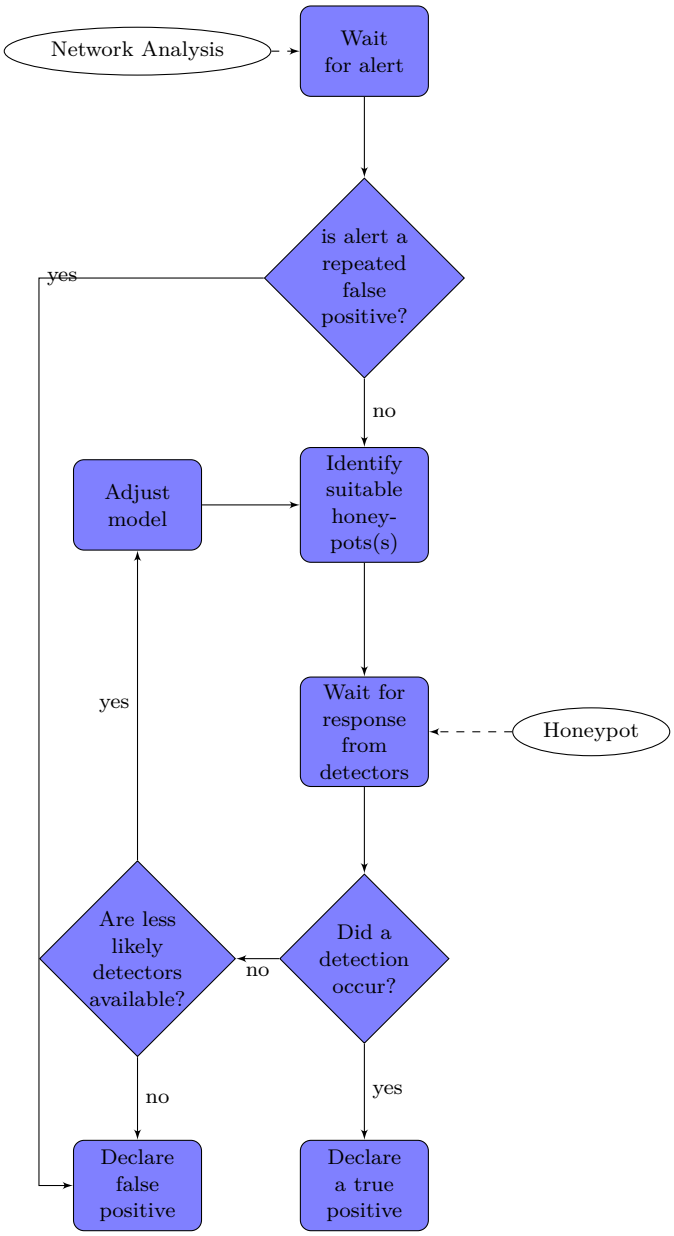

Figure 3.3: Decision diagram for the hybrid detection system

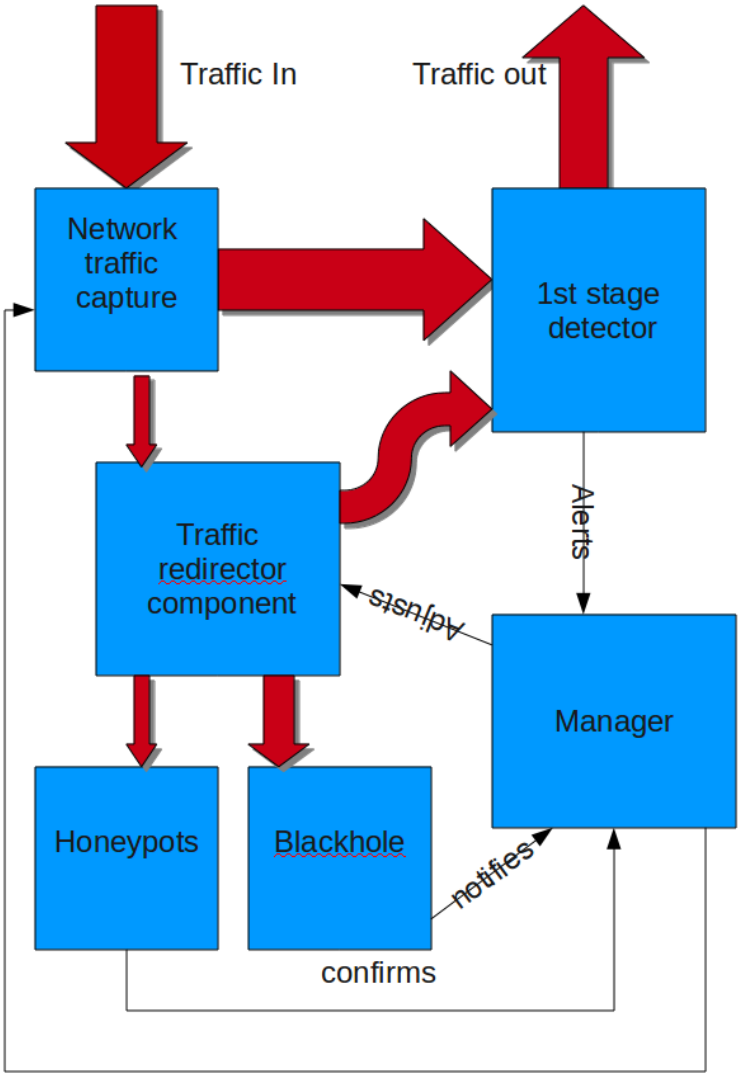

Figure 3.4: System breakdown in components. Red arrows indicating traffic flows and the amount in relation to each other, black lines indicate the (important) messages that are exchanged

Messages The component receives messages which set the MAC and/or IP addresses that are to have their traffic redirected to the Traffic redirector component.

Network Traffic
- **Ingress** Network traffic collected from one or more points in the network.
- **Egress to First stage detectors** The component outputs a stream of network traffic for analysis by network detectors. Depending on the setup this can be live or duplicated network traffic.
- **Egress to Traffic redirector component** The component outputs a stream of traffic originating from and destined for any nodes under investigation to the traffic redirection component. This must be live traffic data and preferably must comprise of all the traffic flowing to and from nodes under investigation.

3.2.2 Traffic redirector component

This component is the second step in filtering and rewriting traffic so it can be routed to second stage detectors. It receives traffic originating from a possibly infected node from the Network traffic capture component. To redirect it to second stage detectors changes must be made to the transport layers and in some instances to application layers to ensure no connections are broken. Furthermore traffic is separated in malicious and non malicious parts to (attempt to) avoid routing non-malicious traffic into second stage detectors. This improves the performance of the system and connectivity of nodes under investigation. We summarized its functions below:

- Filter out any traffic that is unlikely to be malicious and return this to the network.
- Rewrite the transport and application layer of the remaining traffic using reverse address translation and proxy techniques so it can be routed to second stage detectors or black holes.

Redirection occurs based on destination ports and IP addresses. Once a destination IP is mapped to a detector this mapping stays the same until it is reset. To which detector (and if) connections are mapped to is based on the redirection strategy used, see Section 3.3.3. Additional proxy techniques may be used for application protocols that either do not cope well with Network Address Translation(Nat) or which contain identifiable information such as IP addresses or certificates.

Input/Outputs

Messages For each (suspected) infected node a policy is provided that specifies the redirection rules. These redirection rules include specifying to which detectors traffic needs to be mapped and what portion of traffic is to be redirected.

Network traffic
- **Ingress** The input network traffic consists of the traffic intercepted by the Network traffic capture component.
- **Egress to the network** Traffic that does not need to be redirected per the aforementioned policy is returned to the network and will be otherwise unaffected.
- **Egress to Second stage detectors** The remaining (suspected) malicious traffic will have destination IP addresses mapped to addresses of select honeypot detectors or black hole ranges.

3.2.3 First stage detectors Network-based anomaly detectors

The first detection stage in the hybrid system are network-based anomaly detectors. One or more anomaly detectors monitor the network traffic for signs of malicious traffic which indicates a node is infected with malware. Indicators are attacks that are aimed at modifying a victim in some way or actions that abuse or scan for services. Some traffic related to malware is of less interest because these do not cause a measurable change on the honeypot, or because simply no honeypot/host-based detector exists. For example traffic such as such as command an control messages or when malware is posting sensitive information to websites is of less interest.

The goal of first stage detectors is:

- Monitor for malicious traffic indicative of malware infections. The detector should raise alerts on impending or ongoing attacks, abuse or scanning behaviour.

Type of detector The nature of detected malicious behaviour is affected by the chosen anomaly detector. For example some detectors focus on memory corruption attacks where others are more oriented towards detecting script injection attacks on web based systems. This preference towards particular attacks is useful when determining a suitable second stage detector.

Input/Output

Messages Anomaly systems raise alerts containing a source IP address, and optionally a destination port(s) of the observed malicious connection(s). These alerts are sent to the Managing component.

Network Traffic
- **Ingress** Input network traffic consists of the traffic aggregated by the Network traffic capture component.
- **Egress** Depending on how traffic is captured and made available to the network detector(s) traffic will either be discarded or returned to the network.

3.2.4 Second stage detectors

The second stage detectors are detection systems which interact with (suspected) malicious traffic. Therefore a detector must provide a network service for a malicious connection to abuse or attack. The way in which the detection of malicious activity is achieved should be accurate and immediate. Honeypots conform to these properties as attacks or abuse can be detected when these are ongoing or shortly after these have occurred. Not all approaches share these traits however. Detecting brute force attacks on authentication mechanisms relies on counting the number of times and the time spacing of individual password guesses, and web application honeypots typically will just apply rule based detection mechanisms.

These systems are all setup such that these listen for incoming connections on the detectors IP address. Depending on the type of detector it may also be possible to integrate multiple detection systems into one detector, for example by running a high interaction honeypot that applies taint analysis, file system analysis and also provides an open mail relay monitoring for spam.

We discuss the selection of second stage detectors in the section on the detection strategy in 3.3.

- A second stage detector provides a large attack surface, which is monitored for any attacks or abuse.
- Minimally a detector must provide some way of notifying that some malicious act was performed. Ideally it should be able to correlate this to a specific IP address that is performing a malicious act. Other preferred options are the ability to correlate different malicious acts to the attacking IP addresses if these occur simultaneously. Also any parameters relevant to an observed attack such as the type of the attack, any code snippets or captured malware samples should be captured and stored.

Input/Output

Messages Every detector reports detected attacks to the managing component by notifying which IP address is launching attacks.

Network traffic
- **Ingress** A second stage detector will receive some of the suspected malicious traffic from the Traffic redirector component. Recall that the traffic is also distributed between available detectors.

– **Egress** Technically none, any traffic routed to the second stage detectors will connect to it so can not be routed further. Honeypot detectors themselves are allowed to initiate connections to other systems and are not filtered.

3.2.5 Black hole

Rerouting all scans and attacks to honeypots may generate high loads on both the attacking computer and the honeypot as malware suddenly find itself with an unlimited supply of potential targets. To avoid wasting resources once attacks have been detected traffic should be forwarded to lightweight monitors like Darknets or tar pitsThese systems still afford some primitive monitoring of the malware behaviour without generating excessive load on quarantined nodes and second stage detectors.

Input/Output

Messages Reports on the number and frequency of connection attempts to ports.

Network Traffic **Ingress** Receives traffic from the traffic separator, all traffic received originates from systems that successfully triggered a detection by a second stage detector.

3.2.6 Manager

The Manager component serves as the central controller in the system that controls the behaviour and monitors the output of any process in the system. The decision tree that it implements is depicted in Figure 3.3, the way decisions are made is detailed in the next section on the Detection strategy.

It fills two main roles:

- It selects the nodes which need to be monitored using second stage detectors and which detectors are most suitable to achieve a detection. For these decision it can utilize current and historical data obtained from first and second stage detectors and the properties of the attacking node.
- It serves as a data logger and correlates alerts received from the various detectors as to track which alerts yielded confirmations and by which combination of detectors this was achieved.

The manager receives and sends the following messages:

Input/Output

Messages received from

- **First stage detectors** Alerts for nodes emitting malicious traffic containing the details of the connections that caused the alert.
- **Second stage detectors** Alerts from honeypots or other second stage detectors regarding attacks and the attack details
- **Blackhole** Information on connection attempts and their frequency. Used for bookkeeping.
- **Traffic separator** Information on which destination IP addresses have been rerouted, to which detector and the number of connections, used for bookkeeping.

Messages sent to

- **First stage detectors** Results of the second stage detectors can be used to update anomaly models of first stage detectors, so we send the results of a confirmed detection to the first stage detector.
- **Second stage detectors** Some detectors may need to be periodically reset to a new state or started on a per node basis which requires some control messages to be sent.
- **Traffic separator** When a detection is confirmed or a node should no longer have traffic rerouted the traffic separator need to adjust its firewall rules.
- **Network traffic capture** The traffic capture system must receive the MAC or IP address of nodes whose traffic is to be diverted to second stage detectors.

3.3 Detection strategy

There are three decisions that are part of the Detection strategy:

1. When to initiate (or continue) monitoring with second stage detectors, i.e. the decision on whether an alert from the first stage is cause for further inspection using honeypots.
2. When to declare an initial alert a false positive, this means concluding when no detection by a second stage detector occurs that the initial alert(s) are false positives, or reversely that the second stage detector was not suitable for achieving detection.
3. Which second stage detectors to utilize to achieve a detection and how traffic is to be routed to these detectors.

3.3.1 Initiating monitoring

To decide whether to (continue to) monitor traffic means making a decision based on an alert received from a network detector and historical data. Our approach is to start redirecting traffic when an alert is received and to group any subsequent alerts for the same node with the initial alert. If a false positive is declared all grouped alerts are declared to be false positives and any subsequent similar alerts are ignored until a timer expires. The goal here is to avoid permanently redirecting based on alerts that never yield any results. This is exploitable by an attacker by triggering first stage detectors and avoiding triggering second stage detectors until a false positive is declared. Then for the duration of the timer the attacker is free from the system. This attack is somewhat mitigated by the fact that an attacker can not predict the state the hybrid detector is in and what portion of its traffic is redirected or monitored, or when the timer will expire.

3.3.2 Declaring false positives

If traffic is redirected to one or more honeypots but a detection by these honeypots does not occur we must assume either of the following to be true:

- The honeypot(s) failed a detection because there was nothing malicious to detect.
- The honeypot(s) failed even though malicious traffic was present, but no suitable detector was used or traffic was not redirected properly. A detector failed to convert a true positive alert from a network detector into a confirmation by a honeypot.

Declaring a false positive is based on a time or connection counter, when the timer or counter elapses we assume no malicious traffic must have been present. Conversion failures are lumped together with false positives since there is no way to distinguish the two at this level. It may be possible that different patterns emerge from a set of connections and alerts from first and second stage detectors, but for now our only option is to ensure a large surface for malware to attack as to minimize conversion errors. The threshold for the counter affects how long nodes will have their traffic redirected to detectors and the time detectors have to monitor for attacks.

3.3.3 Selecting second stage detectors

Selecting detectors to route traffic to starts with selecting and installing these detectors and optimizing these for our purposes. This is the static part of selecting the honeypot detection. When running the system for every alert received from a first stage detector we estimate which honeypot is likely to achieve a detection.

Configuring suitable detectors There are several properties relevant for determining whether a detector can be used in a hybrid detection approach.

- Provide a large attack surface, a larger surface ensures when attacks are made that these will succeed.
- Correlate network traffic to observed attacks, honeypots based on emulation are typically capable of doing this, but not all high-interaction honeypots can correlate an attack with the source the attack came from.
- Reporting capability, some honeypots can provide detailed attack details such as exploit code, malware samples and or other resources associated with an attack.
- The detector must cover some or all of the malicious behaviour the first stage detectors are likely to detect.

High or low To ensure unlikely or unknown attacks (zeroday or 0-day) attacks are detectable high interaction honeypots must be employed. The objective here is to be able to monitor for attacks that target unpublished vulnerabilities or that deploy new vectors of attack that are not detectable with or implemented in emulation based detectors. When selecting different detectors the main goal is to ensure that the combined attack surface of these detectors covers all possible vectors malware attack or abuse.

However this is not achievable as only a small subset of all vulnerable services and attacks can be detected by low interaction honeypots. Attempting to cover all possible permutations and combinations of Operating systems and services is also impractical. However creating high interaction honeypots covering every variation is not required to create a sufficiently large attack surface for malware to attack.

For example a high-interaction honeypot running a flavour of Windows and IIS is likely to come under attack, either because of old attacks still circulating, or new attacks being encouraged because of low biodiversity (availability of similar systems in high numbers). Reversely, a system running Android in an emulator which provides a service using the Apache web server or something similar is not likely to be a target of memory corruption attacks. The web application provided by the web server might well be a target of an attack, but such an attack could also succeed when the same web application is hosted on a Windows/IIS server. So we can reduce the types of detectors used to cover known attack vectors and include systems that are most likely to be targeted by (new) malware as high-interaction honeypots. A selection would at the very least comprise one low interaction honeypot, and we would expect to augment this detector with various Windows revisions, and perhaps these days even some *nix based systems such as Mac OSX, Ubuntu or even mainstream smart phone systems may prove to be a useful addition.

There are also instances where it may be hard to predict whether a service itself is the target of the attack or the application that runs on top of the service. For example (automatically) distinguishing between an attack that attacks a flaw in the crypto decoding library of a web server from an attack that tries to inject script code. Since its simply not possible to run every permutation of services, web applications and operating systems as high-interaction honeypots some trade offs must be made that reduce the number of honeypots while maintaining a large attack surface. This means that one detector might be able to monitor for web based attacks, and a different detector can monitor for the memory corruption attack. This requires distributing indistinguishable attack traffic to both these detectors.

Attack	Technique	Specific Network Service/OS/other	Implementation	Generic detector
Code injection	Memory corruption	specific service and OS	Argus, Dionaea	no(some)
Brute force	Evaluate long word lists	specific service any OS	Fail2ban	yes
Generic SQL injection	Abuse of poor data checks	any service and OS	glasjvnost?	yes
Specific SQL injection	abuse via custom SQL crafting	specific SQL service	unknown	no
Spamming	abuse of mail servers	any service and OS	Spamassasin	yes

Table 3.1: Some attack types, used techniques and whether these target specific services/systems. Also shown are some approaches that detect the attack or abusive behaviour. For some attacks it is not possible to define one generic detector capable of detecting all variations of that attack type.

Identifying suitable honeypots Having selected a set of detectors that cover a large attack surface the second issue is selecting which to use when monitoring for malicious traffic. The amount of attacks launched by a node is limited by the rate of which it launches attacks and the counters used for declaring false positives. This means the total number of attacks is finite and we therefore must select a strategy that selects an appropriate detector before all chances of detecting an attacks are exhausted.

There are some combinations of first and second stage detectors possible that may never yield a result, such as combining web application honeypots with an anomaly based detector monitoring for scanning behaviour. Frequently indexing services are used to locate vulnerable web applications, which avoids triggering the anomaly detector and never leads to a result. We would expect similar results if we replace the anomaly detector with a detector aimed monitoring for the presence of shell code.

In both these cases the chance of attacks occurring or being launched in that particular fashion, scan first and then attack, is unlikely. This because better attack vectors are available or because an attack is considered too difficult. Indexing services will find vulnerable web applications faster then an attacker can scan these out, and affords a stealthier attack. Remote code execution attacks through memory corruption on web services are also unlikely attack scenarios. Also the population of web server systems is quite diverse and

modern web servers are typically automatically patched and hardened. Most attacks on web servers instead prefer to focus on easier means of exploitation such as weak or poorly secured authentication mechanisms, poorly coded web applications and loose security settings.

Selecting the most likely detector is asking to guess the type and nature of an attack before an attack is detected. This seems impossible, but we can make some guesses based on a few bits of information. The IP address and operating system of the attacker, the destination port, the first stage detector that raised the alert, whether the attacked service is also present on the attacking node and the type of service that is attacked all govern which type of attack is most likely being executed. For example the presence of the attacked service on the attackers side may be indicative of worm like behaviour suggesting a honeypot mimicking the attackers operating system is also vulnerable. The gravitation of exploitation approaches towards the same techniques and vulnerabilities means that by observing the targeted service and the type of detector that certain attacks types are more likely then others. For example attacks on port 22 or the SSH service are nearly always attacks which brute force login credentials.

Then there are also niche attacks to consider such as attacks on vulnerable network stacks or firmware of computing devices. A recent example is a vulnerability in the Windows 2008 TCP/IP network stack implementation [67], or a flaw in firmware of a network interface card [70]. The latter can be ignored as it is too tightly coupled to specific hardware to ever be considered for use in large scale automated attacks. with network stacks there have been issues in the past, but issues with network stacks occur infrequently.

To select the most likely detector to route suspected attack traffic to we assign for each first stage detector a number of probable second stage detectors for each of the various high profile network services or ports. These probable detectors are selected to cover the bulk of known attacks and will receive a large portion of redirected network traffic. In a live deployment we would also include high-interaction honeypots which receive a low fraction of traffic and are randomly selected from a pool of detectors.

Adjusting the model If detectors with a high rating consistently fail to detect anything we can demote these to a lower rating in favour of raising lower rated high-interaction detectors that perform better to a high rating.

redirecting traffic There are two strategies for routing traffic to detectors:

- Route traffic to different detectors sequentially , one detector at a time, until a detection occurs or a false positive is declared in which case a new detector is selected and traffic is routed to this new detector.

- Route traffic to different detectors simultaneously where traffic is distributed between the available detectors based on a predetermined ratio. To avoid confusing malware connections to a destination IP address are always mapped to the same detector.

The latter approach is attractive because we assume malware will attack or abuse more then one node. This means we can distribute the same type of attack to multiple detectors which should ensure a shorter time to achieve detection and/or using more detectors without needing malware to launch more attacks.

We also have to consider what portion of the total traffic from a node we will reroute to detectors. The objective here is to separate non-malicious traffic from the malicious traffic without limiting the attack capabilities of malware. Too stringent filtering here could result in never observing attacks at the second stage detectors because these are never routed to detectors, or because attacks are never initiated at all. For example botnet malware will require a connection to its externally hosted C& C infrastructure before it is likely to launch any kind of attacks.

- Reroute all traffic.
- Reroute only if connections for a flagged destination port occur and reroute any subsequent traffic with the same destination IP.
- Use white listing where only traffic to specific services and/or destinations is ignored.
- Use intermediate IDS to separate attack traffic, for example to attempt separating user generated traffic from attacks on web application/servers. This approach of using network detectors for ad hoc rerouting of traffic into detectors is demonstrated by Anagnostakis et al in [2]. It may seem a bit odd to reapply an IDS at this point, but the only relevant operational parameter here is the false negative rate. This reverses the role of the IDS as we are not interesting in what portion of traffic is actually malicious, but which portion of traffic definitely is not malicious.

We opted to reroute if the destination ports for which network detectors generated alerts are contacted, this appears to be a sensible default balancing accuracy and impact on network. It means if alerts from the first stage detectors correspond to connections that contain attacks we avoid rerouting traffic before malware is launching attacks, and ensure that any subsequent connections that may be required for the attack to succeed are also (re)routed to the same detector.

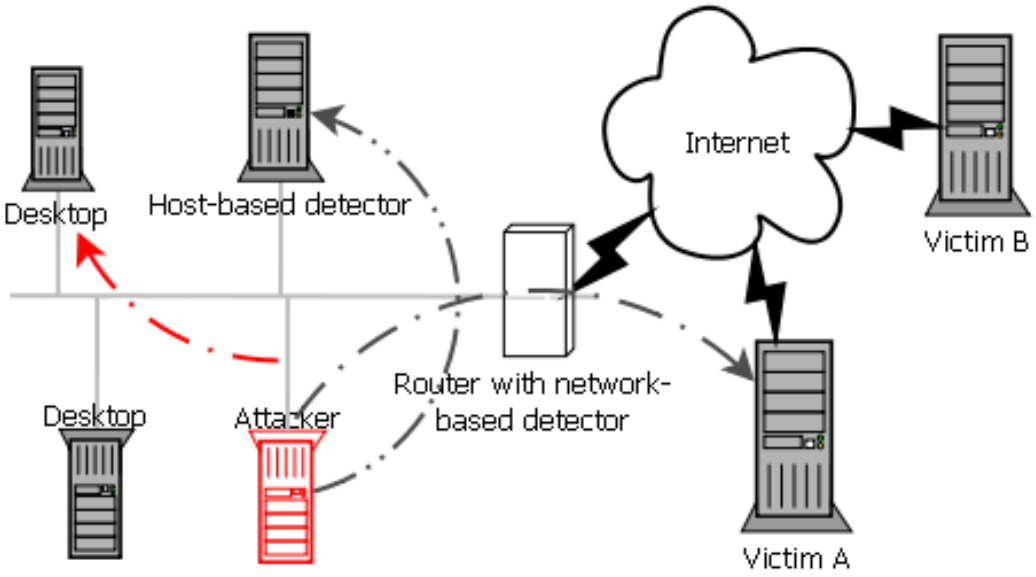

Figure 3.5: The problem of intra network detection, nodes directly connected to another node launching attacks, indicated by a red line are unprotected and attacks may remain undetected until a directly connected host-based detector is attacked. Attacks from or on Internet connected nodes must always traverse a router which allows inspection of the traffic for signs of attacks.

3.3.4 Sensor placement and Mac spoofing

In a deployment the placement of the redirection and network detection components governs what part of the traffic on the network can be observed. To be able to monitor or redirect traffic it must flow through these components. For instance communications between two nodes on a LAN will not traverse a detector placed at the networks edge, seen depicted in Figure 3.5. In the case of intra-net monitoring this means tapping traffic from and redirecting traffic at every switch in the network.

It is not uncommon for malware to only select targets on the local network instead of randomly selecting targets on the Internet. A hybrid of this also exists where malware focuses on specific (geographical) net blocks. In terms of detection both the hybrid way and the random target selection will not be limited to a local network and can be monitored for at the network edge.

To ensure detection of malware that targets both external and internal host the system must monitor traffic traversing the gateway as well as traffic between (at least some) nodes on the local network. So there are two scenarios possible :

- Monitor at (only one of) the network edges, traffic traversing this edge or the network switches attached to it can be monitored. The latter is

achieved via promiscuous mode of network interface cards or by using a tap point on the switch.

- Traffic is tapped at every switch providing network analysis a full view of the traffic flow on the local network.

Providing network analysis with an integral view of the traffic flowing in and out of every node should allow a network detector to detect an infected node and possibly be more accurate then a detector that only views a subset of all the traffic. The trade-off is the increased volume of traffic to be processed, a problem which may only be partly solvable by applying more hardware. To determine if this approach is possible requires measuring the sustained volume of traffic a detector can process. What is important is that a detector eventually is able to monitor at least some traffic so that when a node is infected malicious traffic can be flagged at some point. It is not required that all malicious traffic emitted by a node is detectable via network analysis, provided that eventually an alert is raised by part of the traffic.

Modern Ethernet switches allow extensive control over network flows separating the actual flows from the physical links using Virtual LAN. For example this allows creating disjoint networks using the same cables and network switches or perform network access control. It also allows us to capture traffic from specific nodes for inspection and redirection to a honeypot. In cases where using network management is not practical methods that hack the Ethernet link can be used. These attacks are known as Mac spoofing where ARP tables on switches or on network nodes are manipulated (poisoned) such that traffic is sent to a different MAC address. The downside of some of these techniques are the volume of ARP packets needed to maintain the redirection, particularly when not the node itself but the network switch ARP tables are being poisoned.

Chapter 4

Evaluation

4.1 Introduction

To evaluate our architecture we built a prototype that implements the approach outlined in the previous chapter. Our staged detection approach is a reactive system, which means the approach to monitoring network traffic is changed based on events generated by preceding network traffic.

This, and the fact that detectors directly interact with the traffic, means we can not utilize a typical off-line approach to evaluating this type of systems. Thus we also built a sand-boxed environment to we recreate attacks using both live Malware samples and simulated attacks using standard hacking tools such as Metasploit[65]. The implementation of our prototype and lab is discussed below.

The purpose of this prototype is to study the concept of our approach in general. In particularly we are interested in investigating the following:

- Evaluate if and how coupling honeypots with anomaly-based network detectors is a viable approach towards monitoring for (attacking) malware.
- Experiment with various attack scenarios in a (simulated to be) real environment and relate these with existing approaches such as Snort or single running honeypots.
- Investigate how the various type of attacks (expected to be) launched by malware are detected by honeypots and assess the practicality of using honeypots for detecting such attacks.
- Study the accuracy of detection, can we glean from the experiments anything regarding accuracy of a real system.

In table 4.1 we list some use cases of malicious behaviour we aim to detect and how we expect behaviour to be detectable. We use these to perform a qualitative analysis of how the system functions.

Scenario	Attack summary	Exhibited network anomaly	Exhibited behaviour on honeypot	How to detect
Brute force scanners	Poorly monitored systems using weak authentication or authentication credentials are scanned out and compromised by trying a large number of often reused authentication credentials	Higher then average number of failed connections and higher then average number of successful connections to the same port and IP address combination.	Application protocol interactions are limited to just the first stages, IE so called banner grab is performed. Also high number of failed authentications occur	Locate sources that are likely to be scanning using TRW and redirect traffic for these ports to a high interaction honeypot that can monitor the authentication failures or the subsequent post exploitation behaviour by using easily brute forced credentials.
Malware distribution via direct attacks	Malware is distributed by scanning for and injecting malicious code into vulnerable or poorly secured services	Attacks on services enumerate vulnerable services by scanning and/or directly sending attack payloads to services.	Remote code execution and malware downloads to the honeypot	Scanning behaviour can be detected using TRW both low and high interaction honeypots for second stage detectors
Malware distribution via indirect attacks	Attacks on systems providing services, where the attack turns the attacked service into a service that attacks its own clients, typically without the service itself being affected	Vulnerable services are scanned for or found via indexing services. In the case of HTTP services large amount of probes are to be expected which discover vulnerabilities and web services in use	On the honeypot changes to the network services are made via injection, file uploads and injection of various script code into available web applications	Web based honeypots for detection and anomaly detection based on HTTP GET and POST parameters.
Spam	Sending Spam is one of the functions typically present in malware, and sometimes serves not just for generating income but also distribution of malware	Frequent SMTP traffic that avoids using the local relay servers and or scanning for open relays	There is no behaviour to observe on the honeypot, its just expected to relay traffic.	Detection relies on the performance of the Spam filter in use on a honeypot, and the accuracy with which Spam traffic can be detected.

Table 4.1: Comparison of malware detection and capture systems

4.2 Prototype implementation

In this section we detail how we constructed our laboratory sandbox and prototype implementation.

4.2.1 Sandbox

The sandbox is a VMware ESX system which contains two virtual networks

- DMZ network, This network contains second stage detectors and utility machines. The hosts on the DMZ are not reachable by the network that is monitored or the Internet unless traffic is explicitly routed to these machines by the detector. The detectors themselves can initiate connections to the Internet, this to not disrupt any malware attacks that need to load additional resources.

- Simulation network, This network is monitored using our detection system and to which we connect infected machines or run attack simulations on. It connects to the Internet via the gateway which hosts first stage detectors and also implements a number of filters and redirects in order to keep malware under control.

The simulation networks gateway filters outbound network connections to prevent or limiting attacks on regular systems. However, not all outbound traffic is filtered as malware must be allowed some connectivity to the Internet, as samples may simply not function otherwise. This is an important issue as the functioning and execution of malware code is more and more intertwined with the availability of network services and data. For example, some of our samples can easily mistaken for benign software simply because the network domains these sample connect to do no longer exist. We settled upon allowing HTTP traffic and high order ports, this still leaves some attack capability but also minimally disrupts any essential communications. We apply rate limiting on outbound connections to limit the attack capability of samples that are being analysed.

Both the gateway machines as well as the machines that host the various honeypots are based on Ubuntu-server 10.10 Maverick. To safely access the DMZ and internal simulation network a VPN was created that connected directly to the DMZ zone. We continuously monitored traffic on the DMZ gateway using Tcpdump by simply filtering all the known destinations such as the VPN clients and the IP addresses used for updating Ubuntu and Windows systems. This reduced the data captured by such a degree that we could safely operate malware samples while keeping a sharp eye on any traffic that was leaked to the Internet. A thing to note when utilizing virtualized environments is that the ESX implementation itself filtered traffic based on MAC addresses as a security feature, which needs to be explicitly turned off if one wants to tap or manipulate traffic that traverses a virtual switch.

4.2.2 Hybrid Detector

The implementation of the detector comes down to:

- Selecting network intrusion detectors and honeypot implementations.
- Writing a implementation for the managing of these detectors and the traffic redirection component.

Because our test environment contains just one virtualized network switch, we did not need to explicitly implement a way of capturing traffic. The traffic redirection and first stage detectors both run on the simulation network gateway which also hosts the logic for the manager component and web service

that reports on the status of the detector. We integrated the manager, redirecting component and reporting functionality into one package that runs on the simulation network gateway. Detectors can leverage an XML-RPC interface to interact with the managing component which itself exposes web pages to control the hybrid detector. The choice of using XMLRCP was purely a practical consideration, each detector or honeypot is different in the way data is extracted, so rather then trying to writing an interface for each individual detector we opted for wrapping the detector with XLMRPC.

Manager

The manager maintains the state of the system, tracks received alerts and keeps track of the current redirection table. The latter is needed because without implementing an interface to the Netfilter kernel extensions ourselves there is no easy way to query the Netfilter firewall rules. We also encountered some other limitations with the Iptables implementation that we had to work around, we will discuss these in the next subsection.

The data the manager maintains is structured as a Host entity which can be set to three states which affect how we process alerts:

- Ignore mode : In this mode we do not respond to alerts or confirmations that are received from detectors
- Redirect mode : In this mode any alerts received from the first stage detectors is stored and processed. If this is an alert for a new port we add redirection rules, otherwise we just update the times occurred counter for this alert.
- Drop mode : We engage this mode after a pre-set amount of time has elapsed since the last confirmation was added or initial alert was raised. For all ports for which alerts were received the redirects are removed and drop rules put in place.

From redirect mode the system can transit back to ignore mode, or to drop mode depending on whether confirmations occur. This process is depicted in Figure 4.1.

State	**Alert**	**Confirmation**	**Redirection changes made**
Ignoring	Ignored	Ignored	Never
Redirecting	Added	Added	For new ports
Dropping	Ignored	Added	Never

Table 4.2: Description of the way alerts and confirmations are recorded, confirmations may still occur when the system switches from redirect to drop mode and are therefore recorded in drop mode as well.

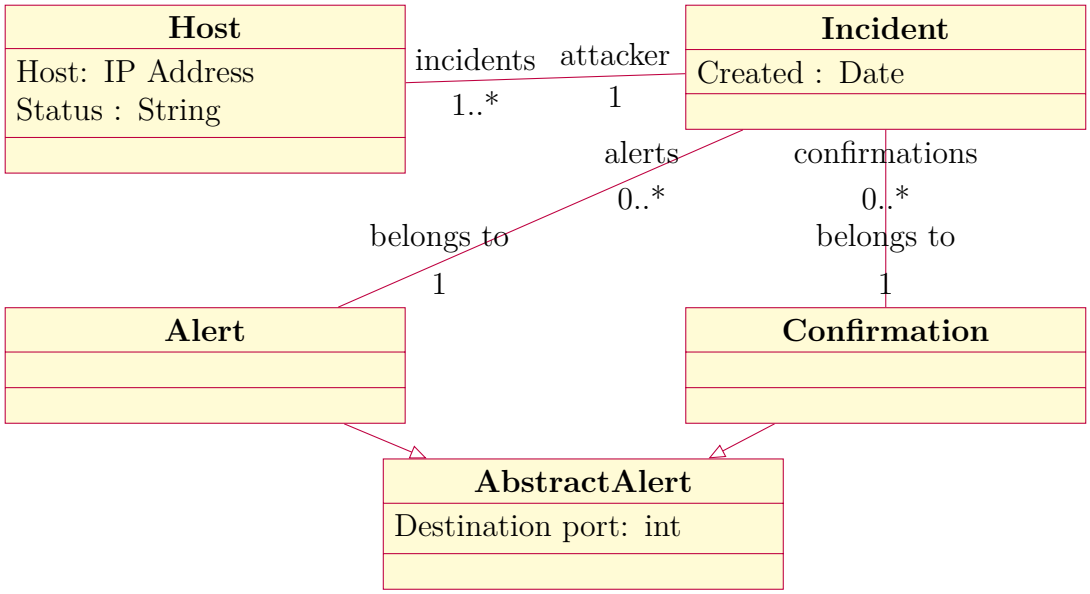

Figure 4.2: Decision diagram for the hybrid detection system

Outside of the Redirection mode we drop all first stage alerts for a host, we could cache alerts while in ignore mode but there seems little incentive to do so, in table 4.2 we list for each state how we process alerts . An issue here is that the first stage detectors themselves might not repeat the same alert for a given amount of time, so if we switch mode just after we ignored an alert it will take up to the expire time of the detector before the alert is repeated. In drop mode we will still add confirmations instead of dropping them as these may still occur just after changing modes due to timing issues.

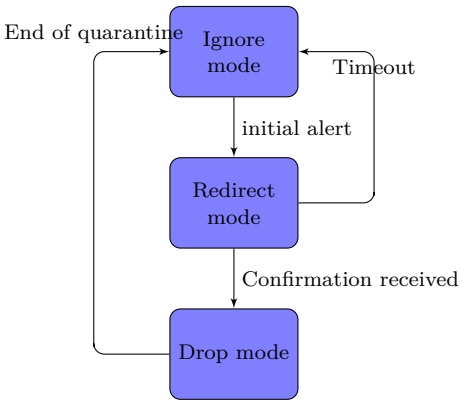

Figure 4.1: State transition diagram for the hybrid detection system

An incident is an abstraction that we use to track alerts and confirmations for a host from the moment that a host is placed in redirect mode until it is placed in ignore mode. Figure 4.2.2 depicts how these data types are related. To the current incident for a host we add alerts and optionally confirmations. If an alert is reported for an unknown host we create the new host, add to it an incident and place the host in redirection mode. The most recent incident is the current incident until replaced by a new incident. This allows us to keep historic data on past alerts.

Redirection component

Redirecting traffic is performed using the Netfilter implementation in the kernel which we manipulate using Iptables. We outlined a number of properties that we intended to utilize for distributing traffic between second stage detectors, but have not been able to implement all of these using the Iptables interface. The two main limitations we encountered were:

- Setting the percentage of traffic that is directed to each detector. The problem we encountered was that while we could randomly match a rule and set it to trigger only after connections to certain ports were observed we could not persist this mapping. So instead of sending traffic to the same detector as the initial connection subsequent connections would be distributed between the detectors as set by the ratio that we specified. The work around in this case was to resort to fixing the ratio by not using the random module of Iptables which was able to persist mappings, at the cost of selecting the ratio.

- Redirecting traffic without resorting to masking source IP addresses. Because we directly apply DNAT filtering on two networks that are directly linked through the gateway the system would try to route returning connections instead of applying the network address translation again. So connections were successfully rerouted but then responses would be dropped by the gateway (because they could not be routed) instead of having the remote address replaced by the original IP address via the NAT tables. We have not been able to find a practical solution using Iptables to solve this other then applying source masquerading.

Aside from these limitations we were successful in creating firewall rules that allows us to dynamically distribute traffic evenly between various detectors. The latter issue limits us to only monitoring one host on the test network as we have to hard code the IP address. However as far as we can tell these issues are related to the way the Iptables and routing is implemented, but not made impossible by the TCP/UDP protocols.

Blackhole component In our test system we are not particularly concerned with monitoring what happens when we are with monitoring a host, so we just drop traffic to ports for which an alert exists and at least once confirmation is associated with the incident.

First stage detectors

As first stage detectors we installed the Bro[32] and Snort [41] Intrusion detection systems which monitored the network interface attached to the simulation network. We utilized the default snort rules as distributed with the snort package. From the Bro IDS we used the TRW implementation which was slightly

modified to track scanner IP address and destination port tuples instead of just source IP addresses. Other emerging IDS implementations are Suricata [72] but we did not integrate this detector.

Second stage detectors

We installed a number of honey-pots and host based detectors to use for second stage detectors

- Dionaea, a low interaction honeypot, designed to capture direct attacks via remote code injection using Libemu[63], and can also capture files via FTP or SMB services. It itself allows registering handlers for events generated within the honeypot, we added a handler that submits a confirmation when code injection is detected.
- Argos, a high interaction honeypot within which we installed a Windows XP OS (SP2). It reports code injection attacks either directly via a terminal or a control socket and also dumps log files of these events. We had difficulty correlating the attacking IP address to the code injection reported by the virtual machine, but from what we gather from available literature this is possible. Because we are forced to masquerade the source IP address we did not further investigate this issue.
- Fail2Ban, technically not a honeypot but a script that parses the system log files looking for the brute-forcing of login credentials. We added a script so it reports directly reports any login attempts it considers brute-force attempts.
- SMTP mail server, to provide a SMTP service for sending spam we use a short python script that implements some mail server functions. Beyond reporting that mail is submitted it performs no other functions.
- GlastopfNG, which is a web-based honeypot aimed at detecting automated attacks and various forms of script injection.

4.2.3 Traffic capture

We experimented with redirecting of traffic at the Ethernet level using the Ettercap and Scapy libraries. This did not give any issues during testing aside from the Scapy library being slow and dropping packages. The tool Ettercap did function as was expected. We did not perform any kind of load testing with these programs.

4.3 Experiments

We experiment with our prototype by evaluating the use cases described at the start of this chapter, either by simulating the use case or by using live malware samples which exhibit behaviour that fits a use case. The purpose of these experiments is to evaluate what kind of attacks or abuse can be detected and also evaluate how suited individual honeypot implementations are for this type of use.

Examples of detection We provide several examples of malware executions and simulations by a system which is monitored by our hybrid detector below. Each execution is allowed to continue for several minutes and we record detections by first and second stage detectors. Where appropriate we also note why behaviour is detectable and where it is not. The results of these experiments are summarized in Table 4.4 and Table 4.4 at the end of this chapter. Simulation of some behaviour is necessary due to limited availability of the number of samples which generate malicious traffic when executed. Samples which do not immediately generate malicious traffic may still generate traffic at a later point in time, which could be monitored for by allowing samples to run for longer intervals or even schedule periodic execution of a sample. However we manually monitor the execution of malware samples so doing so is not practical, instead we opted for simulations as a substitute. The

Sasser The Sasser worm is a typical Internet worm which aggressively scans for targets to attack. It is an example of malware performing a direct attack using remote scanning to find targets. Because this is an old worm honeypot detectors based on the Windows XP SP2 OS and an Argos detector is not vulnerable to this worm, but a low interaction honeypot like DIONAEA is capable of detecting the attacks this worm launches. This worm is predictable in its behaviour which makes it very useful for experimenting.

This worm performs remote scanning and direct attacks, which should be detectable using the TRW algorithm used as a first stage detector. The worm first pings a host before before directly attacking, the attack directly triggers a generic snort rule for NOP sleds and also the first stage TRW detector after several failed connection attempts. In some instances the sample launches very few attacks which exposed a problem with the way we currently distribute connections to the various honeypots. If the number of connections is too low, due to the way IPTABLES seems to balance connections, only the first detector will receive connections. When the worm attacks alerts are also raised for port 9995 which is used to distribute the worm binary. Additionally the Ports 138 and 137 are flagged for scanning due to UDP broadcast packets.

Blaster The blaster worm sample is a example of malware performing direct attacks and uses local scanning to find its targets. It utilizes ARP scanning

to localize targets and launches attacks on the windows DCOM service on port 135 which downloads and executes the worm binary. We do not monitor ARP requests with TRW but if there are enough hosts on the local network without a service on port 135 it is detectable by the first stage detector using TRW, so we simulated with 3 such hosts on the local network. The attack was quickly identified by both the network detector and the low-interaction honeypot DIONAEA.

Netsky The Netsky worm is an email worm which attempts to send copies of itself to any email addresses it can scrape from an infected machine. We observed it attempting to (only) use Yahoo SMTP servers to send spam messages, which makes it an example of a hit list attack. Since SMTP connections are blocked from leaving the simulation network these trigger the first stage detector and mail is sent instead to a simple SMTP daemon which received some spam emails. The second stage detector is not a real honeypot as it is only capable of accepting email but can not distinguish Spam from normal email messages. For this a traditional spam classifier would be needed.

Credentials brute forcing Using the Metasploit toolkit we simulated a remote scanner performing SSH account brute forcing. This is an example of the brute-forcing of credentials use case using random scanning. We expected our system to easily detect this type of attack and behaviour but on several attempts the scanning phase failed to trigger the TRW detector and so connections were never redirected. It remains unclear why this occurred and only occurs in a few instances.

MS08-67 simulation To test the Argos honeypot and simulate detecting a modern remote scanning worm such as the Conficker worm we used the Metasploit scanner tool to scan for the vulnerable port and manually launched some attacks on some of the detected services, which (all) are the honeypot detectors once the TRW detector is triggered. Manually launching attacks caused the honeypot detectors to post confirmations to the hybrid detector system.

HTTP bot (Waledac) The HTTP bot malware we evaluate is of the Waledac variety, it did not initiate any other connections beyond HTTP requests to no longer existing web-servers. Since this type of infrastructure is commonly in use by modern malware we evaluate several similar binaries of the Zeus malware variant with similar results, though in some instances HTTP connections to existing web servers known as drop zones are observed. We expected to observe the downloading of addition malware binaries and/or the launching of attacks or spam, but did not observe this behaviour. This could be due to these samples already being published on a public malware tracker, lack of new command instructions or infrastructure not being available. The

net detection result here is a false negative for all samples. Sometimes the web requests to non existing infrastructure leads to alerts by the first stage detector, but these never lead to detections by a second stage. Since these are still malware samples collecting personal information which is submitted to a server we consider this malicious traffic which is not detected as such.

P2P bot A different type of botnet is a P2P based botnet binary, this binary does not submit or request information via web services but utilizes peer 2 peer networking to exchange information. This malware did not launch any attacks but it does generate significant amounts of UDP traffic which causes the first stage detector to raise a large number of alerts, we recorded over 100 different ports being flagged as malicious. Given that for none of those ports a second detector is available all of those would be ignored leading to a false negative on detection. During the five minutes we let the sample run we did not observe it performing any other network activity beyond the UDP based traffic.

Bittorrent To simulate a use case of a legitimate program triggering network detectors we downloaded a Linux distribution using Bittorrent, which resulted in the high order port typically used by Bittorrent to be flagged for scanning, as well as ports 80 and 53. Port 80 seems to be flagged because of multiple HTTP requests to services providing peer lists which do not respond in time. The only unexpected port to be flagged is port 53/UDP used for DNS queries as we redirect all DNS queries to a local DNS server, and so queries always receive a response. We attribute this to the Bro detector as this behaviour also occurs with other UDP services like those on ports 138 and 137.

4.3.1 Limitations of the experiment

For three of our use cases we are able to recreate or simulate the described behaviour. For the the indirect attack case no malware samples exhibited that type of behaviour, and extensive simulation is made superfluous by the lack of a honeypot with the facilities to detect these indirect attacks adequately. We are however able to evaluate the direct attack and Spam use cases as well as simulate the brute forcing of credentials behaviour use cases.

The shortage of malware samples exhibiting certain behaviour is a result of the way modern malware or botnet malware is integrated with and the dynamics relation of malware and online resources. The result of this dynamic relation is that malware alters behaviour or simply does not function at all depending on the availability or nature of these resources. Furthermore malware can implement techniques that hamper analysing the malware or change its behaviour when it is executed in a virtual machine or debugger. Finally malware may only exhibit certain behaviour when it is instructed to do so. This makes the use of botnet based malware, at least in our case, impractical. (note Recently work has been published on how to run and simulate your

own botnet within a research environment [9]). The only solution to this is to utilize approaches that allow monitoring malware binaries over longer time intervals.

4.3.2 Comparison with single honeypot

Malware target selection can be divided into four general groups of behaviour. The local and remote scanning behaviour randomly selects targets to contact on or off the local network. The opposite of this behaviour is the use of a hit list for nodes to attack which distinguishes from the scanning approaches by a much lower failed connection ratio. A special case of the hit list is where only a single note is (repeatedly) attacked. We compare the number of connections required before these attack scenarios are detectable using honeypots and network based detectors in Table 4.3.2. In this comparison we assumed a local network size of 255, and a global network size of 4B. The single honeypot monitors one IP address whereas the network detectors monitor the local sub net.

Target selection	Single honeypot	Hybrid detector
Local scanning	(1-255) 128	(5-∞) 5
Remote scanning	(1-4B) 2B	(5-∞) 5
Hit list target	∞	(5-∞) ∞
Single target	∞	(5-∞) ∞

Table 4.3: (Best case - Worst case) and average number of malicious connection required before detection occurs. Payload based detection applies signature- or anomaly-based analysis and assumes if an initial connection does not trigger the detector subsequent (similar) connections will not either. The TRW detector counts connection attempts as well as connections.

4.4 Analysis of results

Having implemented a prototype and experimented with the various use cases we conclude that redirecting to honeypot detectors as a means of detecting malware infections is a viable, but not very practical approach. The reason for these have to do with the context that attacks expect to be executed in, or the lack for want of a specific context, and concerns regarding accuracy. We will discuss these below and briefly revisit these issues and our initial research questions at the end of this thesis.

Context It is definitely possible to glean from network traffic whether a particular machine is infected, but in the cases not involving direct attacks by

remote code injection utilizing a host based detector presents no advantage, and in some cases will be detrimental towards detection.

Spam and credential brute forcing We found that in cases where the context is less relevant, such as various authentication protocol interaction we can only redirect if no authentication of endpoints takes place, in which case transparently proxying connections rather then redirecting allows the same level for monitoring connections. In the case where context is relevant, say someone is attempting to brute force a Google Gmail account we lack the context to asses whether this is taking place. In such an instance it would be more useful to monitor the HTTP response headers or page content. The same example applies to detecting Spam, where spam is sent via other SMTP servers the network stream can be observed, and samples carved directly from the stream or the connections proxied. If a web based form is abused a detector needed to detect such abuse is not present, and could not be detected unless traffic is redirected to an exact duplicate of the abused web form instrumented for measuring Spam abuse.

Indirect attacks Redirecting indirect attack traffic is very likely to be detrimental to detecting malware infections rather then improving detection results. Detectors exist for detecting generic forms of web attacks, but all these fail to detect targeted attack that take into account the context of an application or the way it integrates with other services. Reversely with the intensive (ab)use of social networks and protection systems present on those same social networks it is very likely that attacks can not be observed until malware is logged in to the social networking site. For example there are protections which serve to prevent attackers from abusing vulnerabilities to perform cross site request forging attacks (CSRF). However these same protections also prevent malware from, for example, directly posting a malicious URL to a social networking site if it were to use CSRF countermeasures. To post an URL the malware would first have to go through the login process and obtain the CSRF token used as a protection against CSRF attacks. This ties detecting the malware attack to a specific sequence of actions and data that it expects, which means without the context of the web site present the attack is unlikely to be executed. Note that the malware could just as easily steal the CSRF token from, however what constitutes the token is dependent on the context of the application. So the malware could still only attack if the service its communicating with is the original intended target, or a honeypot that approximates the same context.

Direct attacks Remote code injection attacks on honeypot systems or other direct attacks that upload malicious files also depend on context. However the current situation is that a large portion of Internet connect system runs Windows XP and the majority of malware is, currently, targeting this

platform. Even though in the future this might change we argue that the range of contexts we would need to present for malware to succeed remains limited to major patch levels such as service packs. So whereas the number of different contexts for web applications is infinite the range of operating systems or patch levels is not. This means that for every suspected direct attack there is a limited set of contexts in the form of high interaction honeypots to evaluate which makes redirecting traffic a viable strategy.

Accuracy Since traffic is redirected any errors that are made are costly, though only slightly less worse then dropping traffic. However the honey-pots can only report an attack has taken place, if no attack is reported it is unclear whether the alert is a false positive or the attack is not detected(or unsuccessfully because of lacking context). Given that we can not definitively conclude that a true positive detection by the first stage always yield to a confirmation by the second stage, can this approach be more accurate then the detector deployed by itself? It seems that this is only true if retuning a first stage detector for a higher trade-off in false positives and true positives is possible, and the increase in detection rate is offset by the conversion rate of the second stage detectors. This is not immediately obvious to be the case, nor did we find any useful metric describing this relation between detection ratio for network-based detectors, or were able to establish a statistically significant conversion ratio from our own system. What we can say about this approach is even with short redirection intervals it is likely that any intrusion detection approach tuned more sensitive then what it is designed for yields a disproportionate higher increase in false positive and thus connections that are, if however brief, blocked. So we do not expect an improvement off detection rates by network-based detectors that already perform well.

Use case	Sample	Target selection	Detected by (1st/2nd)	Result	Notes
Direct attack	Sasser	Remote scanning	TRW/DIONAEA	TP	very noisy
Direct attack	Blaster	Local scanning	TRW/DIONAEA	TP	
Spam/Indirect	Netksy	Hit list	TRW/MAILTRAP	TP	detectable because blocked SMTP connections triggering TRW
Bruteforcing	Scanner Credentials	Simulation	TRW/FAIL2BAN	TP	
Direct attack	Scanner MS08-67	Simulation	TRW/ARGOS	TP	
Indirect attack/Spam	Waledac	Hit list	TRW/No attack	FN	Only Fetched configuration settings
Indirect attack/Spam	P2P bot (Gimmev or Grum)	Unknown	None/No detector	FN	P2P high order port UDP traffic to Ru IP addresses, file sharing
Not malware	Bittorrent	Remote scanning	TRW/No detector	TN	DHT generates many failed connections

Table 4.4: List of evaluated use cases, the samples used and the way these select targets to attack. In the detected by column we list the detectors that raised any alerts, details for these alerts are in Table 4.4. No Attack or No detector means no attack is observed or that there is no second stage detector for the (suspected) malicious traffic. In the Result column we rate the result or failure of detection as True or False detections. The indirect attack examples are rated False Negatives(FN) because even though no attack s observable other (not covered by the use cases in Table 4.1) malicious traffic is still present.

Sample	1st stage	2nd stage	Notes
Sasser	138,137,445,9995	445	138,137 not part of attack
Blaster	137,138,138		138,137 not part of attack
Netksy	25,53,123,137,138,445	25	137,138,445 not part of attack
Scanner Credentials	22	22	
Scanner MS08-67	445	445	
HTTP bot (Waledac)	80	None	no attack observed
P2P bot	100+ high order ports	None	no attack observed
Bittorrent	53,80,139,6789	None	not malware

Table 4.5: Alerts raised for various ports and confirmations by second stage detector for each experiment

Chapter 5

Conclusions

5.1 Revisit research questions

- Given that honeypots are near perfect classifiers of malicious behaviour can honeypots be extended to have a broader network view by actively re-routing suspicious traffic to the honeypot detector?

Yes, in principle any system designed to function as a honeypot can have traffic routed or redirected to it (whether based on indications provided by anomaly-based network detectors or not). This allows constructing detection systems using anomaly detectors and honeypots that are both resilient against attacks on the detectors itself and capable of detecting known and unknown attacks. We have shown this by designing and prototyping a detector that combine these detectors. However for most of our uses cases we discovered that using detectors in this fashion is not a practical approach for any use case bar direct attacks as for the other approaches network analysis can achieve similar or better results without blocking or redirecting connections.

- Re-routing traffic into honeypot detectors has a detrimental effect on network performance as a whole, can we formulate a strategy that minimizes routing non-malicious traffic into honeypots?

Our design incorporates a strategy that based on the assumption of malware launching multiple attacks distributes suspected attack traffic between likely detectors, and only diverts traffic to these detectors if connections to suspected attack ports are initiated. This may break some attacks where the initial connection to a different port is not flagged by a detector, but we are not aware of any attacks that would be hampered by this approach. We were able to recreate and simulate attacks which the system detected using worm malware samples and simulating behaviour using the Metasploit tool-kit, except for the indirect attack case where we resorted to published analysis of malware samples to evaluate how well we can expect these to be detected.

5.1.1 Honeypots as network detectors

The first question we wanted to answer is whether we can detect malware this way, and how well this functions. We had three worm samples that performed direct code injection or SMTP spamming attacks and also simulated SSH brute forcing and code injection using the Metasploit toolkit. Even though we distribute traffic these attacks did not seem to suffer any negative effects and in all cases we observed near instant results as soon as the anomaly detector raised alerts for ports. We also acquired samples of the recent Conficker and Stuxnet worms, but neither would function in our VM. We do know however that the Conficker worm is detectable using both the low and high interaction honeypot we deployed, so given that it also exhibits scanning behaviour should be detectable. The Stuxnet worm is a less clear case. As we understand it, this malware does not actively scan but utilize information from its host computer to locate new victims. Because of this behaviour it already has some information about its victim prior to initiating a connection, which entails that a redirection strategy such as used in our approach is not viable.

5.1.2 How does it compare

Honeypots on a local network are likely to detect malware that scans the local network, or uses broadcast information to locate new victims. If the honeypot does not broadcast traffic or is not explicitly listed somewhere then it is unlikely to get attacked by malware that does not scan for targets (for example hit list malware). Our experiments seemed to confirm this behaviour, the worms that scanned the local network were instantly detectable by a local honeypot, whereas a sample that randomly selected a target IP address was not. During the experiments we continuously ran Snort, but it did not pick up any of the attacks from the older worms, presumably because the samples were too old.

5.1.3 Suitable detectors

Do all honeypot approaches make suitable detectors? In our description of the implementation we listed a number of approaches that we used or implemented to be used as second stage detectors.What we found when using these detectors is that the context a detector provides, within which attacks can be/are executed, is typically limited. If the number of different possible contexts is large, or even see seemingly infinite such as the variation in available web applications, detectors employ rule based approaches to achieve detection instead of the traditional model where the honeypot itself is monitored for changes.

This distinction is important because the reliance on rules also means the specific context that a real honeypot can provide is not used or available. Compare for example rule based SQL injection monitoring with a detector that monitors a real web applications database for query structures that should not

be occurring. Any injection attack that relies on features present in this web application for its attack, or perhaps requires to be logged in to a certain web page, will be much harder to detect, if detectable at all with a rule based approach that does not provide the same context as the web application. This issue of lacking context is less of an issue with direct attacks because here the number of different contexts is small, and in the case of brute forcing (services) or Spam non existent.

So monitoring traffic for indirect attacks by redirecting to detectors may be worse then monitoring the interaction of attacks with services. We do not argue that the implementation of the particular web based honeypot that we used is lacking, but that it and any other rule based implementations can not emulate the context of the web application that is attacked. However it is clear to us that without context being a factor there is no advantage of utilizing a host based detector over a network based detector.

So we make the argument that for the purpose of detecting indirect attacks via web applications it does not appear to be desirable to redirect traffic for the purpose of detecting attacks. Correlation-based approaches which track (differences in) inputs and outputs to web applications and so externally monitor for changes to behaviour, for example Atlantides[7], seem far more suitable approaches. Perhaps such approaches can be augmented with client side honeypots which monitor applications suspected of having been attacked by malware and are suspected of now spreading malware themselves. For credential brute-forcing and Spam the reliance on rules is less of an issue as in these instances the distinction between good and bad behaviour is relatively well defined and so ways of evading these rules are limited. To be precise these use cases depend on the protocols used instead of individual implementations of these protocols.

5.1.4 Making errors

The fourth and final point of interest when experimenting was the conversion ratio, i.e. when did alerts lead to confirmations. We observed that alerts for the various worms that we execute in our sandbox yielded immediate results, as soon as traffic was redirected based on an alert we would see results. This would suggest that keeping the interval until a false positive is declared short a good approach. Even though there is not enough data on how often a real attack would fail to be confirmed we can show an example where an alert would fail, where it will not be confirmed even though a legit attack took place. A vulnerability the Stuxnet malware exploits to distribute copies of itself is the MS10-061[68] vulnerability. We tried attacking a standard Windows XP SP2 honeypot with this attack, and this gave no results. This is expected because the attack makes use of a particular configuration setting that leaves the system vulnerable. This could be solved by adding more high interaction honeypots or weakening the existing honeypot but it seems impossible to cover every

operating system and permutation of security settings.

5.1.5 Future research

We propose the following future research:

- Staged network-based detection, rather then redirect traffic to honeypots use alerts to preselect (future) traffic for analysis. Several approaches towards network-based detection are attractive because of high accuracy and inherent ability to detect unknown attacks, but see limited deployment because of limited processing capacity. For example network level shell code emulation is a very attractive approach for detecting direct memory corruption attacks, but is not sufficiently fast enough to be applied on a large network. However if traffic were to be pre-filtered for malicious traffic using lightweight anomaly based detectors and so reduced to a manageable set size this may prove to eliminate or reduce the limitation of processing capacity. This proposal relates to our project because it applies the same two stage approach towards detection, but now without the penalties imposed by blocking connections. Also related to this is the application for ad hoc interception of encrypted (but unauthenticated) traffic so a second stage is able to apply content based analysis. This is a somewhat controversial approach, but acceptable within a local network given certain conditions are met. This is a relevant consideration because with the imminent (and unavoidable) large scale adoption of IPv6 coupled with pervasive support for hardware accelerated strong cryptographic support we may well see increased use of encrypted (but unauthenticated) network connections. Which would make large scale use of content based analysis difficult.

- Staged client side honeypot, Some indirect web-based attacks aim to alter or probe web pages. In cases where pages are altered to point to malicious code these changes may be detectable using statics analysis or high interaction client honeypots. So we propose to utilize existing approaches that can already provide indicators of possible attack behaviour and use these for generating leads of possibly malicious web-pages, and subsequently analyse these pages using static/dynamic analysis approaches. Also a reverse approach may be interesting, if a machine is suspected of launching attacks it itself may provide services or listen on odd ports.

- Baiting malware, Instead of diverting traffic to honeypot detectors and risking costly mistakes by (unavoidably) blocking legitimate connections we propose to decouple the detectors by no longer diverting traffic to detectors. Instead network based analysis is used to determine which hosts may be utilizing external data sources to locate targets to attack and detectors are dynamically added to these sources. This entails that if a node queries a target list this will include a honeypot detector. For

example Stuxnet appears to utilize context information from its victim to locate other systems to attack. This list is constructed by information obtained via broadcast and peer information exchanges. Here we avoid the negative side effects of making costly mistakes, while still actively searching for malware within the network. It is unclear how well this approach would extend beyond the SMB protocol, but is nevertheless interesting approach as malware abusing the SMB protocol to attack peer systems or distribute copies is a popular technique.

Recommendations

One of our recommendations is to improve the availability of malware samples, specifically for making available samples for which the Internet infrastructure is still active. While obtaining malware samples for analysis is trivial we found that for most of our samples network infrastructure was no longer present. While this is unavoidable we even experienced this issue with samples that at the time of analysis were approximately two to three weeks old. Also analysing malware behaviour, particular the network traffic generated by malware is so far only analysed on an ad hoc basis. Several binary analysis approaches that continuously analyze and sample new malware exist such as Bitblaze[42][59], Anubis[56] and Joebox[62], but so far limited attention has been given to analysing and classifying network traffic on a similar structural level.

5.2 Conclusion

In this thesis we proposed the use of honeypot detectors as network detectors and designed, prototyped and evaluated a scheme that aims to extend a honeypot into a full fledged network detector monitoring for signs of malware infections. From our results and experiences gained when prototyping and experimenting we conclude that while no technical objections exists this approach remains hampered by decreased accuracy as a result of failed detections by honeypots, and decreased connectivity as a result of false positives by first stage network detectors. Note that the alternative would be to not filter traffic at all, or block connections outright and with that accept blocking legitimate traffic as well.

The core of the problem is matching alerts for potential attacks from network detectors to a proper second stage detector capable of detecting the attack, either via emulation or because it provides the appropriate context in the form of a vulnerable service and operating system. This is most evident with the indirect attacks use case as there are many different web applications which can be attacked, and not all of these web based services can be sufficiently emulated or recreated on a local honeypot detector. Detecting direct attacks is limited in the same way, but here a disproportionate amount of attacks focus on a small subset of service implementations, operating systems

and vulnerabilities, which allows using a smaller set of detectors while still being able to detect the majority of attacks.

However this means that it seems impossible to apply our approach as a generic detector because:

- Too many different second stage detectors means we can not ensure that suspected traffic is rerouted to detectors until detection occurs as we can only reroute traffic for a limited amount of time.
- It is not feasible to recreate a suitable target context for every conceivable (indirect) attack because the targeted web service does not lend itself for recreating inside a detector, for example because the target of an attack is behind some form of authentication. This limits the ability to recreate the targeted context within a detector. For Network services and operating systems this also applies but due to lower diversity the impact is also less often felt.

The problem with using honeypots as network detectors is that the set of detectable behaviour by a honeypot and first stage detectors are divergent. An algorithm like TRW can not distinguish between the various use cases, which is why we devised a scheme that allows combining several honeypot detectors which together cover the use cases. However as we've seen its only partly possible to create second stage detectors which together cover the behaviour detected by a first stage detector. As a result second stage detectors will always fail to detect some attacks that are flagged by first stage detectors, which seems a problem which is not easily solved.

Bibliography

[1] Hyang ah Kim. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security '04: Proc. 13th Usenix Security Symposium*, pages 271–286. USENIX, 2004.

[2] K.G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A.D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Security '05: Proceedings. 14th USENIX Security Symposium*, pages 129–144, 2005.

[3] S. Antonatos, K. Anagnostakis, and E. Markatos. Honey@home: a new approach to large-scale threat monitoring. In *WORM 07:. Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 38–45. ACM, 2007.

[4] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. In *RAID '06: Proc. 9th International Symposium on Recent Advances in Intrusion Detection*, pages 165–184. Springer, 2006.

[5] M. Bailey, E. Cooke, F. Jahanian, A. Myrick, and S. Sinha. Practical darknet measurement. In *ICICS '06: Proc. 40th Annual Conference on Information Sciences and Systems*, pages 1496–1501, 2006.

[6] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, C. Kruegel, and UC Santa Barbara. A view on current malware behaviors. In *LEET '09: Proc. 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats, 6th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2009.

[7] D. Bolzoni, B. Crispo, and S. Etalle. Atlantides: An architecture for alert verification in network intrusion detection systems. In *LISA '07: Proc. 21st conference on Large Installation System Administration Conference*, pages 141–152. USENIX, 2007.

[8] David Brumley, Li hao Liu, Pongsin Poosankam, and Dawn Song. Design space and analysis of worm defense strategies. In *ASIACCS '06: Proc. 2006 ACM Symposium on Information, Computer, and Communication Security*, pages 125–137. ACM Press, 2006.

[9] J. Calvet, C.R. Davis, J.M. Fernandez, J.Y. Marion, P.L. St-Onge, W. Guizani, P.M. Bureau, and A. Somayaji. The case for in-the-lab botnet experimentation: creating and taking down a 3000-node botnet. In *ACSAC'10: Proc. 26th Annual Computer Security Applications Conference*, pages 141–150. ACM, 2010.

[10] G.F. Cretu, A. Stavrou, M.E. Locasto, S.J. Stolfo, and A.D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *S&P '08: Proc. 28th IEEE Symposium on Security and Privacy*, pages 81–95, 2008.

[11] G.F. Cretu, A. Stavrou, S.J. Stolfo, and A.D. Keromytis. Data sanitization: Improving the forensic utility of anomaly detection systems. In *HotDep '07: Proc. of the 3rd workshop on on Hot Topics in System Dependability, 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,DSN'07*, 2007.

[12] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. Honeystat: Local worm detection using honeypots. In *RAID '04: Proc. 7th International Symposium on Recent Advances in Intrusion Detection*, pages 39–58. Springer, 2004.

[13] L. Ertoz, E. Eilertson, A. Lazarevic, P.N. Tan, V. Kumar, J. Srivastava, and P. Dokas. Minds-minnesota intrusion detection system. Published in Data Mining, Next Generation Challenges and Future Directions, 2004.

[14] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proc. 15th USENIX Security Symposium*, pages 241–256. USENIX Association, 2006.

[15] G. Gu, Z. Chen, P. Porras, and W. Lee. Misleading and defeating importance-scanning malware propagation. In *SecureComm '07: Proc. 3rd International Conference on Security and Privacy in Communication Networks*, 2007.

[16] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security '07: Proc. 16th USENIX Security Symposium*, pages 1–16. USENIX Association, 2007.

[17] X. Jiang and D. Xu. Collapsar: a vm-based architecture for network attack detention center. In *SSYM'04: Proc. 13th conference on USENIX Security Symposium*. USENIX Association, 2004.

[18] S Keemink and M Kleij. Implementing snort into surfids. Master's thesis, Universiteit van Amsterdam, 2008.

[19] O. Kolesnikov and W. Lee. Advanced polymorphic worms: Evading ids by blending in with normal traffic. In *USENIX Security '06: Proc. 15th USENIX Security Symposium*. USENIX Association, 2006.

[20] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[21] J. Li, S. Stafford, and T. Ehrenkranz. Sword: Selfpropagating worm observation and rapid detection. Technical report, University of Oregon, 2006.

[22] Z. Li, M. Sanghi, Y. Chen, M.Y. Kao, B. Chavez, and IL Evanston. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *S&P '06: Proc. 27nd IEEE Symposium on Security and Privacy*, page 15, 2006.

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

[24] M. Mahoney and P.K. Chan. Phad: Packet header anomaly detection for identifying hostile network traffic. Florida Institute of Technology Technical Report CS-2001-04, 2001.

[25] D. Moore, C. Shannon, G. Voelker, and S. Savage. Network telescopes: Technical report. Technical report, CAIDA, 2004.

[26] D. Moore, C. Shannon, G.M. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *INFOCOM '03: Proc. 22th Joint Conference of the IEEE Computer and Communications Societies*, 2003.

[27] Er Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. Spyproxy: Execution-based detection of malicious web content. In *USENIX Security '07: Proc. 16th USENIX Security Symposium*, pages ??–?? USENIX, 2007.

[28] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *RAID '06: Proc. 9th International Symposium on Recent Advances in Intrusion Detection*, volume LNCS, pages 81–105. Springer, 2006.

[29] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *NDSS '05: Network and Distributed System Security Symposium (NDSS)*, 2005.

[30] James Newsome. Polygraph: Automatically generating signatures for polymorphic worms. In *S&P '05: Proc. 25th IEEE Symposium on Security and Privacy*, pages 226–241, 2005.

[31] V. Paxson, S. Staniford, and N. Weaver. How to 0wn the internet in your spare time. In *Security '02: Proc. 11th Usenix Security Symposium*, pages 404–413. Usenix Association, 2002.

[32] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23):2435–2463, 1999.

[33] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *S&P '06: Proc. 27nd IEEE Symposium on Security and Privacy*, 2006.

[34] M. Polychronakis, K.G. Anagnostakis, and E.P. Markatos. Emulation-based detection of non-self-contained polymorphic shellcode. In *RAID '07: Proc. 10th International Symposium on Recent Advances in Intrusion Detection*, volume 4637 of *LNCS*, pages 87–106. Springer, 2007.

[35] M. Polychronakis, K.G. Anagnostakis, and E.P. Markatos. Network-level polymorphic shellcode detection using emulation. *Journal in Computer Virology*, 2(4):257–274, 2007.

[36] M. Polychronakis, K.G. Anagnostakis, and E.P. Markatos. An empirical study of real-world polymorphic code injection attacks. In *LEET '09: 2nd Workshop on Large-Scale Exploits and Emergent Threats, 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, 2009.

[37] M. Polychronakis, P. Mavrommatis, and N. Provos. Ghost turns zombie: exploring the life cycle of web-based malware. In *LEET '08: Proc. 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. USENIX Association, 2008.

[38] G. Portokalidis and H. Bos. Sweetbait: Zero-hour worm detection and containment using low-and high-interaction honeypots. *Computer Networks*, 51(5):1256–1274, 2007.

[39] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *SIGOPS Oper. Syst. Rev.*, 40(4):15–27, 2006.

[40] Niels Provos. A virtual honeypot framework. In *Proc. 13th USENIX Security Symposium*, pages 1–14. USENIX Association, 2004.

[41] M. Roesch. Snort - lightweight intrusion detection for networks. In *LISA '99: Proc. 13th USENIX conference on System administration*, pages 229–238. USENIX Association, 1999.

[42] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. In *ICISS '08: Proc. of the 4th International Conference on Information Systems Security*, Hyderabad, India, December 2008.

[43] E. Stinson and J.C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *WOOT '08: 2nd Workshop on offensive technologies, 17th USENIX Security Symposium (USENIX Security '08)*. USENIX Association, 2008.

[44] Y. Tang and S. Chen. Defending against internet worms: A signature-based approach. In *IEEE INFOCOM '05: Proc. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, 2005.

[45] M. Van Gundy, D. Balzarotti, and G. Vigna. Catch me, if you can: evading network signatures with web-based polymorphic worms. In *WOOT '07: 1nd Workshop on offensive technologies, 16th USENIX Security Symposium (USENIX Security '07)*. USENIX Association, 2007.

[46] S. Venkataraman, A. Blum, and D. Song. Limits of learning-based signature generation with adversaries. In *NDSS '08: Proc. 16th Annual Network & Distributed System Security Symposium*, 2008.

[47] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A.C. Snoeren, G.M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *Operating Systems Review*, 39(5):148–162, 2005.

[48] K. Wang, G. Cretu, and S.J. Stolfo. Anomalous payload-based worm detection and signature generation. In *RAID '05: Proc. 8th International Symposium on Recent Advances in Intrusion Detection*, volume 3858 of *LNCS*, pages 227–246. Springer, 2006.

[49] K. Wang, J.J. Parekh, and S.J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *RAID '06: Proc. 9th International Symposium on Recent Advances in Intrusion Detection*, volume 4219 of *LNCS*, pages 226—-248. Springer, 2006.

[50] K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In *RAID '04: Proc. 7th Symposium on Recent Advances in Intrusion Detection*, LNCS, pages 203–222. Springer, 2004.

[51] Y.M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *NDSS '06: Proc. 13th Annual Network and Distributed System Security Symposium*, 2006.

[52] MM Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *ACSAC '02: Proc. 18th Annual Computer Security Applications Conference*, pages 61–68, 2002.

[53] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *ESORICS '09: Proc. 14th European Symposium on Research in Computer Security*, volume 5789 of *LNCS*, pages 232–249. Springer, 2009.

[54] Qinghua Zhang, Douglas S. Reeves, Peng Ning, and S. Purushothaman Iyer. Analyzing network traffic to detect selfdecrypting exploit code. In *ASIACCS 07: . Proc. of the ACM Symposium on Information, Computer and Communications Security*, pages 4–12. ACM, 2007.

[55] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou. Collecting autonomous spreading malware using high-interaction honeypots. In *ICICS '07: Proc. 9th International Conference Information and Communications Security*, pages 438–451. Springer, 2007.

Web References

[56] Anubis: Analyzing unknown binaries. http://anubis.iseclab.org//.

[57] O. Arkin, F. Yarochkin, and M. Kydyraliev. The present and future of xprobe2. http://boss.mikrozet.wroc.pl/~wiesiek/123/html/scan/next.pdf, 2003.

[58] P. Bania. Evading network-level emulation. Technical report, arXiv.org, http://arxiv.org/abs/0906.1963, 2009.

[59] BitBlaze: Binary analysis for computer security. http://bitblaze.cs.berkeley.edu/.

[60] T. Cymru. The darknet project. http://www.cymru.com/Darknet, 2004.

[61] SANS Institute. The top cyber security risks. Technical report, SANS Institute, sept '09.

[62] Joebox: Analyse your malware on windows simply and easily. http://www.joebox.ch/.

[63] libemu - x86 shellcode emulation. http://libemu.carnivore.it/.

[64] T. Liston. Welcome to my tarpit: The tactical and strategic use of labrea. Technical report, Dshield, 2001.

[65] Metasploit - penetration testing resources. http://www.metasploit.com/.

[66] Will Metcalf and Victor Julien. Snort baitnswitch.

[67] Microsoft. Microsoft security bulletin ms09-048 - critical vulnerabilities in windows tcp/ip could allow remote code execution (. Technical report, Microsoft, September 2009.

[68] Microsoft security bulletin ms10-061 - critical
vulnerability in print spooler service could allow remote code execution (2347290). http://www.microsoft.com/technet/security/bulletin/ms10-061.mspx.

[69] The Honeynet project. Sebek. https://projects.honeynet.org/sebek/.

[70] SecurityTracker. Input validation flaw in intel pro/1000 linux drivers lets remote users deny service and potentially bypass security controls. Technical report, SecurityGlobal.net LLC, 2010.

[71] L. Spitzner. Know your enemy: Genii honeynets. Technical report, The Honeynet Project, http://old.honeynet.org/papers/gen2/, 2005.

[72] Suricata. http://www.openinfosecfoundation.org/.

[73] Jack Whitsitt. The bait and switch honeypot: An active and aggressive part of your network security infrastructure., 2003.

[74] G. Wicherski. Medium interaction honeypots. http://pixelhouse.net/midinthp.pdf, 2006.