http://www.wombat-project.eu

European Commission
Seventh Framework Programme
Theme ICT-1-1.4 (Secure, dependable and trusted infrastructures)

ICT-216026-WOMBAT

Worldwide Observatory of Malicious Behaviors and Attack Threats

Analysis of the state-of-the-art

Workpackage	WP2 - Analysis of State of the Art and Requirements
Deliverable	D03 (D2.2)
Author	Federico Maggi, Stefano Zanero
Version	0.3
Date of delivery	May 31, 2008
Dissemination level	Public
Responsible	POLIMI
Data included from	FORTH, NASK, FT, EURECOM, VU, HISPASEC, SYMANTEC

Executive Summary
This document contains a detailed analysis of the state-of-the-art tools and research approaches for malware collection and analysis. We have reviewed high-/medium-/low-interaction honeypots and malware collection tools and worldwide initiatives. The analysis of the collected malware is covered by a comprehensive review of the most relevant research proposals, also including techniques that have been used to analyze "running programs" in general, to be adapted for the WOMBAT purposes.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 216026.

Contents

1 Introduction

Worms, viruses, trojans, keyloggers (or "malware" more in general) are a constant threat for our Internet activities. Day by day, more instances and variants of malware appears, discouraging people from using Internet at its full potential. There is a consensus of opinion among security experts that combating cyber-crime becomes harder and harder. Recently, experts from major anti-virus companies have acknowledged the fact that the cyber-crime scene is becoming increasingly more organized and more consolidated [184, 170, 218, 40].

There are several initiatives (some widely recognized, some known only inside the field), that gather information on this issue. However, such information is not sufficient to the research community to identify, understand and eventually defeat the threats we are facing. The reasons are twofold. First, due to privacy or confidentiality issues, most of these sources are not allowed to share the detailed information they hold. Second, as a result of the lack of publicly available information, no framework exists to rigorously investigate emerging attacks using different data sources and viewpoints.

The WOMBAT project is aimed to address these two issues by building a framework (and offering vetted access to a dataset) that will enable scientific research to be carried out in order to fight more efficiently the ever increasing cyber-crime.

This project is structured around three main objectives:

Data acquisition: We will take advantage of existing sources of information controlled by some of the partners, such as the *Deepsight threat management system* managed by Symantec, the worldwide distributed honeypot system operated by Eurécom, the nation-wide early warning system in use by CERT Polska and the largest malware collection in the world accumulated by Hispasec (thanks to its Virustotal project). However, we will also try to join our efforts with other players in the field as much as possible, and see how their dataset can be used, in order to obtain a global view of the observed phenomena. Also, some new types of sensors will be considered, especially in the domain of client-based honeypots. An important effort will be devoted to ensure interoperability among these various sources.

Data enrichment: Experience shows that the sole observation of a phenomenon does not suffice to reveal its cause(s). Other elements surrounding or characterizing it must be formalized and taken into account. For instance, the observation of thousands of different malware is one thing. Identifying the fact that they are variants of a limited number of different originating worms is another one. Figuring out, based on detailed analysis of the code, that these are likely to come from an even more limited number of groups of authors is another important element that could enable us to understand the kind of enemies we are facing. In the context of the project, we will develop new techniques to characterize the observed attacks, the collected malware, etc. This will lead to the semi-automatic generation of metadata associated with the raw data collected.

Threats Analysis: We will build upon the recognized expertise of several partners in correlating the data and metadata related to various events in order to identify their cause(s). Note that this is very different from correlating alerts for intrusion detection purposes. Indeed, we

are interested in finding the *root* cause(s) of a *group of intrusions* rather than single ones. As a result, we will not only be able to raise alerts more accurately when new situations emerge but, more importantly, we will offer support during the decision making process for countermeasures selection. Last but not least, these models will help security actors to derive sound rationales for their security investments.

This document is a a critical assessment of the current state of the art in these research areas, based on partners' knowledge, literature, product research and external cooperation. There is an obvious preference towards experimentation and use of open source and/or open standard solutions, which we will be able to use freely in the next steps of the project. This evaluation is conducted also for the monitoring projects and tools run by the consortium participants.

For each technology or system we provide a short description, references, and a first-order evaluation in terms of techniques, goals, position impact, importance, strengths and weaknesses. This analysis has been carried out from the perspective of the aforementioned objectives of the project, with the final aim to decide which technologies or existing systems can be used as building blocks for WOMBAT. In addition, in the conclusions we identify the technology gaps we intend to fill.

The document is structured into three chapters. In Chapter 2 a taxonomical survey of honeypots is presented including the most relevant tools available along with some proposed approaches in this research area; this chapter also includes a critical review of malware collection systems and attack detectors. In Chapter 3 we analyze the existing data collection infrastructure such as Internet telescopes, distributed honeypots and worldwide malware/log collection initiatives. Chapter 4 is a comprehensive review of the state-of-the-art in the field of code analysis with focus on malware code analysis techniques along with signature generation proposals.

2 Building block tools

2.1 Honeypot systems and honeynets

Honeypots are a valuable tool for gathering and analyzing information concerning cyber-attacks. They are a widely used as a large information source for malware activities. The concept of honeypot dates back to 1995. In [87], the authors introduce the idea of "intrusion deflection" to attract attackers towards "a specially prepared, controlled environment for observation". This concept has been formalized by L. Spitzner in [182] through the following definition: *A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.* Another definition is: "trap set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems" [13].

There are both server-side and client-side honeypots. Most of the implementations of server-side honeypot solutions emulate the presence of network hosts on a set of unused IP addresses (also known as *dark space*). Honeypots are non-production systems, which means machines that do not belong to any user or run publicly available services. Instead, in most cases, they passively wait for attackers to attack them. The usage of unassigned IP addresses allows to filter out benign traffic and focus on the malicious one, since no benign user is supposed to be contacting these hosts. On the other hand, this biases the type of malicious network activities observable by these honeypots: only untargeted attacks blindly scanning an IP range comprising the honeypot address will be detected by the honeypot. This has two important consequences:

- An attack scanning a portion of IP space different from that to which a honeypot belongs is undetectable to that honeypot.
- More sophisticated targeted attacks aiming at the exploitation of a *specific* network resource considered valuable to the attacker are not visible to honeypots.

These limitations must be taken into account when exploiting server-side honeypots to collect data on Internet network attacks.

Client-side honeypots are a specific, and novel, type of honeypot, which we describe better in Section 2.1.4. Honeypots can also take other forms, like files, database records or e-mails (often called honeytokens). We do not take them into account in this report.

Honeypot implementations also differ in the solutions adopted to emulate the presence of a network host in a network. Spitzner [182] breaks down the different solutions according to the *level of interaction* with the attacker, talking about *low-interaction* and *high-interaction* solutions. During the course of years a new category of "medium" interaction honeypot was also added. Each type presents several good characteristics but also a number of drawbacks; therefore, each type is suitable for specific problems, and it complements the other types. In the following subsections, a detailed overview of honeypot categories is provided, as well as the respective systems that have been developed.

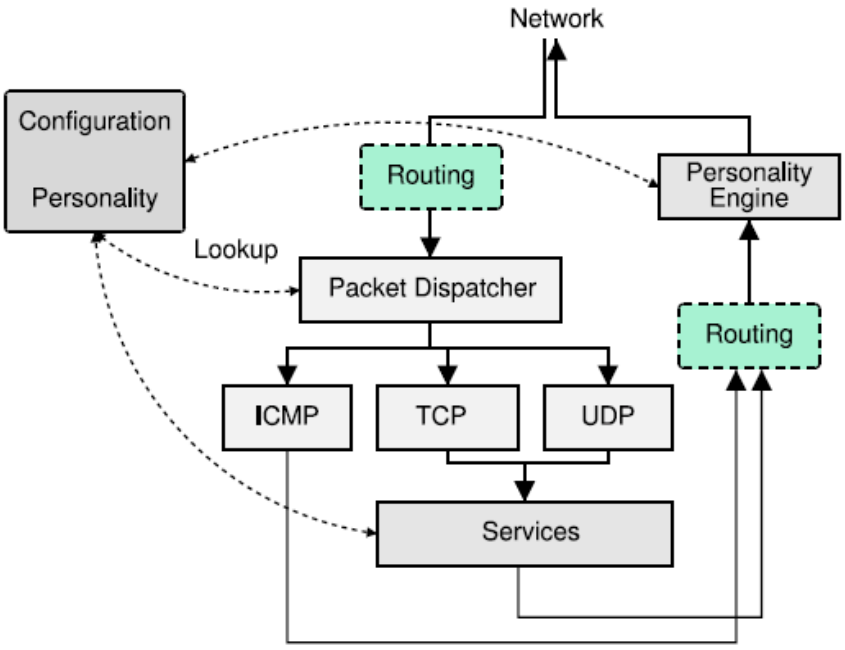

Figure 2.1: Architecture of honeyd

2.1.1 Low-interaction honeypots

A low-interaction honeypot tries to emulate real services and features of operating systems, usually through specialized components designed to mimic real software, albeit with some limitations. The degree with which a low interaction honeypot emulates a network host can vary significantly among different honeypot implementations. The different *depth* of these solutions offers different perspectives on the observed attacks, generating a trade-off between the complexity of the solution and the richness of the collected information.

We can find in the literature simple low-interaction implementations that do not aim at emulating protocol interactions. For instance, the Deception Toolkit by Cohen [47] aims at dissuading attackers from hitting a given machine by advertising the fact that the machine is a honeypot. The LaBrea honeypot [113] actively tries to *damage* the attacking clients by stalling their network connections.

The Tiny Honeypot [24] developed by G.Bakos is a very simple example of protocol emulation. In this approach, a single daemon is generated taking advantage of *xinetd* and the various connection attempts are redirected towards the honeypot daemon taking advantage of netfilter rules. Any connection attempt on any port is presented with a login banner and a root shell, in the hope of collecting information on the intent of the attacker.

The main advantage of low-interaction honeypots is that they are very lightweight processes and their emulated services are unlikely to be infected as they are usually scripts or responders that imitate real responses. Their main disadvantage is that emulation is inaccurate and can only detect previously known attacks.

In the following we review some of the most successful low-interaction honeypots.

Honeyd

The most popular low-interaction honeypot is honeyd [161]. Honeyd is in fact a framework for building honeypots. Honeyd is able to create arbitrarily large virtual networks. In order to achieve this, honeyd comes with a number of modules that respond to ARP requests to claim addresses, request addresses from DHCP servers and finally even emulate latency and packet loss characteristics of the network. Virtual topology created with honeyd is configurable. The user can define the range of IP address space that will be handled by honeyd, the number and type of virtual hosts and their communication characteristics. An example of honeyd configuration is as follows

```
set windows personality "Windows XP Home Edition Service Pack 2"
add windows TCP port 80 "scripts/web.sh"
bind 10.1.0.2 windows
```

In the example above, honeyd will claim that at IP address 10.1.0.2 there exists a machine running Windows XP Service Pack 2. Honeyd does not bind to any sockets; instead it performs network stack emulation, thus it is highly scalable and can listen to an arbitrarily large address space.

Honeyd focuses on the emulation of all the layers of the network stack of a virtual host. Honeyd implements a *personality engine*, that modifies every packet generated by the virtual host in order to make it comply with the behavior of a particular TCP/IP stack implementation. A number of OS fingerprinting tools [221, 116] take advantage of the diversities in the implementation of the TCP/IP stack and in the flags being used to detect the nature of a given operating system. Honeyd modifies the generated packets in order to comply with these heuristics and emulate the *personality* of the different virtual hosts. It maintains a database of network stack characteristics, originally created by p0f[220] tool. When it needs to respond on behalf of a virtual host that "runs" Windows XP SP2 packets are formed in such a way to resemble, according to the database, those sent by a normal XP SP2 operating system. This property makes the virtual honeypot resilient to remote OS detection scans. The architecture of Honeyd is shown at Figure 2.1. Once a packet is received, it passes through the packet dispatcher. The dispatcher sends the packet to the appropriate services based on the configuration. Before the response from the service is sent to the network, the personality and configuration engines are invoked to determine the appropriate transfer protocol characteristics.

While the personality engine is in charge of emulating the transport and link layer of the network stack, the emulation of the application protocols is left to a set of external plugins in charge of the emulation of the protocol interaction. Honeyd supports three different techniques to provide emulators for application protocols:

- **Honeyd Service scripts.** It is possible to associate an executable file (eventually a script written, e.g., in bash or perl) to a given port on a given honeypot profile. Whenever a connection is established, an instance of the script is generated receiving the attacker input on standard input. Any output generated by the process is sent back to the attacker. In the example above, the virtual host will have port 80 open and the port will be handled by the "web.sh" script.

- **Python services.** Honeyd provides an interface to generate python modules associated to the emulation of the protocol interaction on a given TCP port following a non-blocking I/O model. These modules are loaded within the honeyd core and do not require the instantiation of additional processes, and are thus more efficient. Each service is associated to a specific port, preventing the emulation of complex services spanning on different ports (e.g. the FTP protocol).

- **Subsystems.** It is possible to execute external Unix applications within the honeypots virtual address space. This is achieved using dynamic library preloading, intercepting the normal network libc calls and replacing them with their a custom honeyd implementation.

Despite the flexibility offered by honeyd in interfacing to emulators for application-level protocols, the availability of good emulation scripts for the different protocols is scarse. It is possible to find in the honeyd website [5] a few scripts that provide basic emulation for ASCII protocols (telnet, IIS services, ...). More complex protocols, such as the NetBios protocols, cannot be easily emulated with Honeyd and require the usage of full-fledged implementations running as subsystems.

Honeytrap

Honeytrap [216], developed by Tillman Werner, is a remarkable low-interaction honeypot for its smart interaction capabilities. Differently from other approaches, honeytrap is not bound a priori to a set of ports. It takes advantage of sniffers or user-space hooks in the netfilter library to detect incoming connections and bind consequently the required socket. Each inbound connection can be handled according to 4 different operation modes:

- **Service emulation.** It is possible to take advantage of responder plugins similarly to what happens with honeyd.
- **Mirror mode.** When enabling mirror mode for a given port, every packet sent by an attacker to that port is simply mirrored back to the attacker. This mode is very functional, and is based on the assumption that, in case of a self-propagating worm, the attacker must be exposed to the same vulnerability that he is trying to exploit.
- **Proxy mode.** Honeytrap allows to proxy all the packets directed to a port or set of ports to another host, such as a high interaction honeypot.
- **Ignore mode.** Used to disable TCP ports that should not be handled by honeytrap.

Honeytrap takes advantage of a set of plugins to exploit the information collected by the network interaction. The ability of these plugins to handle network attacks can be assimilated to a *best effort* service. For instance, if an HTTP URL appears in the network stream, honeytrap will try to download the file located at that URL. If the HTTP URL is not directly present in the network stream, but is embedded within an obfuscated shellcode, honeytrap will not be able to detect it or download it. Honeytrap vision on the network attacks is thus not uniform, but heavily depends on their structure and their complexity.

LaBrea

LaBrea [3] is a program that listens to unused IP address space and answers connection attempts in such way that attackers at the other end get stuck. The original purpose of this tool was to slow down scanning from machines infected by the CodeRed worm. LaBrea claims unused IP address space by responding to ARP requests that remain unanswered (similar to farpd). When a SYN packet is destined for the claimed IP addresses, a SYN-ACK that tarpits the connection attempt is sent back. LaBrea also tries to mimic normal machines but in a limited fashion, e.g. it can respond to ping and send RST to SYN-ACK.

Kojoney

Kojoney [51] is an application-specific low-interaction honeypot that emulates an SSH server process. The program also keeps track of the credentials used by attackers during the log-in attempts. Attacker source addresses are logged and later located to provide basic attack reporting.

Billy Goat

Billy Goat [162] is a peculiar honeypot developed by IBM Zurich Research Labs that focuses on the detection of worm outbreaks in enterprise environments. It is thus called by the authors Worm Detection System (WDS) in opposition to classical Intrusion Detection Systems. Billy Goat automatically binds itself to any unused IP address in a company network, and aims at quickly detecting and identifying the infected machines and retrieve information on the type of activity being performed.

In order to gather as much information as possible on the kind of activity observed by the sensors, Billy Goat employes responders that emulate the application level protocol conversation. While the general responder architecture is very similar to that employed by Honeyd, Billy Goat takes advantage of a peculiar solution for the emulation of SMB protocols, that consists in taking advantage of a hardened version of the open-source implementation of the protocol [12]. Such an implementation is derived from the work done by Overton [143], that takes advantage of a similar technique to collect worms trying to propagate through open SMB shares. This choice puts Billy Goat in a hybrid position between low and high interaction techniques, since it offers to attacking clients the real protocol implementation, even if hardened to reduce security risks. The increased level of interaction of this technique has allowed interesting analyses such as the work done by Zurutuza in [224]. In this work, the author takes advantage of a data mining tool [97] to automatically generate signatures for unknown events observed by the Billy Goat system.

2.1.2 Medium-interaction honeypots

Medium-interaction honeypots also emulate services but, unlike to low-interaction honeypots, they do not manage network stacks and protocols themselves. Instead, they bind on sockets and leave the operating system do the connection management. In contrast with systems like honeyd, which implement network stacks and protocols, they focus more on the application-level emulation part. The most-well known medium-interaction honeypot is nepenthes.

Nepenthes

The nepenthes platform [134] is a system that was designed to automatically collect malware. Historically, it is the evolution of *mwcollect* platform. Its functionality is based on five types of modules: vulnerability, shellcode parsing, fetching, logging and submission modules. Vulnerability modules emulate the vulnerable services, like a DCOM service or a WINS server. Shellcode parsing modules analyze the payload received by vulnerability modules and try to extract information about the propagating malware. If such information is found, fetch modules download the malware from the designated destination and finally the malware is submitted to a central service (disk, database, anti-virus company) through the submission modules. The whole process is logged by the logging modules. For the time being, only sixteen vulnerability modules have been implemented for well-known exploits, like buffer overflow in Microsoft RPC services, buffer overruns in SQL server 2000 and exploits in the LSASS service. Nepenthes was originally designed to capture malware that spreads automatically, like Blaster or Slammer worms who were targeting hosts blindly.

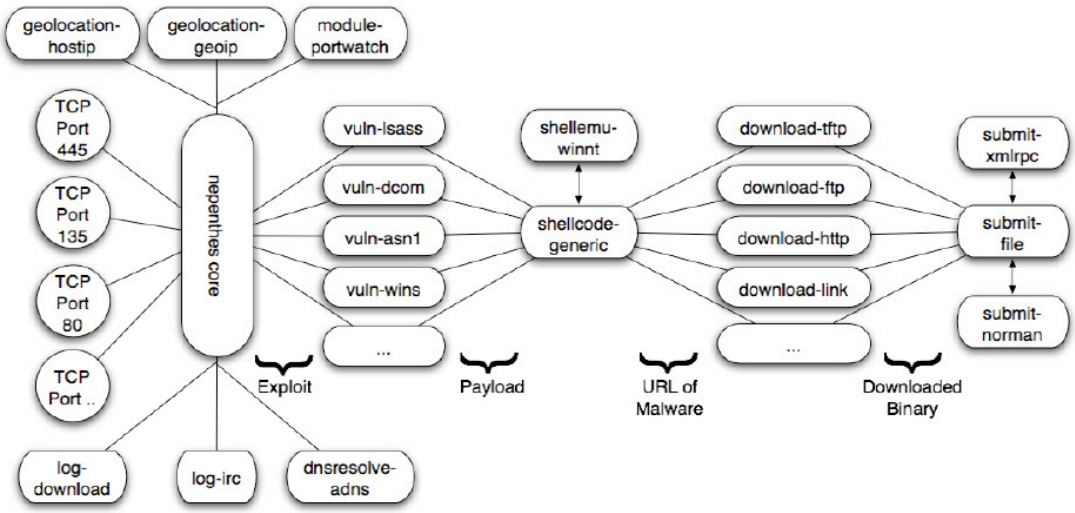

Figure 2.2: Architecture of Nepenthes

The host running nepenthes listens to several ports on one or more black IP addresses. The assignment of these addresses to this host and creation of virtual interfaces in order to have multiple IP addresses to a single interface must be done by the administrator manually. As the host running nepenthes listens to many open ports, it is vulnerable to detection. The workflow of Nepenthes is shown in Figure 2.2. After a connection is established to one of the open ports, the payload of the packets of this connection is handled by the appropriate module. The main restriction here is that for each open port we can only have one vulnerability module. This means that for example we cannot emulate vulnerabilities for both Apache and IIS simultaneously. Vulnerability modules do not provide full service emulation but only emulate the necessary parts of the vulnerable service. When the exploitation attempt has arrived, the shellcode parsing modules analyze the received payload. In most cases, this parsing involves an XOR decoding of the shellcode and then some pattern matching is applied, like searching for URLs or strings like “CreateProcess”. If a URL is detected, fetch modules download the malware from the remote location. These modules implement HTTP, FTP, TFTP and IRC-based downloads. However, a shellcode parser can be more complicated. Some malware can, for example, open command shells and wait for commands or bind to sockets. Shell emulation modules of nepenthes provide command emulation for the virtual shells. Most shell commands are trivial, like echo or START directives.

The limitations of the knowledge-based approach used by Nepenthes have appeared in a recent paper [222] by Zhuge et al. In this paper, the malware collection capabilities of Nepenthes have been compared with those of specially instrumented high interaction honeypots, running the full implementation of an OS. The difference in collection ability of the two approaches is striking: of 171 malware families observed by the high interaction honeypots, only 61 were correctly observed and downloaded by Nepenthes.

The Nepenthes platform has evolved to a distributed network of sensors. Institutions and organizations participate in the mwcollect alliance, where all binaries captured are submitted to a central repository, accessible to all members of the alliance.

Multipot

Multipot [92] is a medium-interaction honeypot for the Windows platforms. Multipot follows the same design as nepenthes. It emulates six vulnerabilities (among them the well-known MyDoom [186] and Beagle [185]). When multipot receives a shellcode, five shellcode handlers try to emulate it. Most common handlers are: `recv_cmd` for shellcodes that bind cmd to a port and receive commands, `recv_file` for shellcodes that open a port and receive a file and finally `generic_url` that performs XOR decryption and extracts HTTP, FTP, TFTP and echo strings. If a location is detected, the malware is downloaded. Multipot sets a maximum download size limit of 3 Megabytes to avoid malware that try to exhaust disk space.

2.1.3 High-interaction Honeypots

High-interaction honeypots, unlike the two previous categories, do not emulate services. On the contrary, they are running services in their native environment. This allows the maximum interactivity with attackers as vulnerabilities are not emulated but actually exploited. Recent advances in virtualization allows the creation of scalable and secure high-interaction honeypots. Operating systems of honeypots are running inside a virtual machine, like VMware [203], Qemu [27] and Xen [25]. This choice was made for two reasons. First, we can run multiple virtual machines in a single physical machine. Thus, we can run hundreds of high-interaction honeypots with a limited number of physical machines. Second, as high-interaction honeypots are vulnerable to being compromised, virtual machines can act as a containment environment. Once the vulnerability is exploited, the honeypot can be used as an attack platform to propagate worms or launch DoS attacks. Instrumented versions of virtual machines can be used as a containment mechanism to prevent vulnerabilities from being exploited and in parallel maintain the highest level of interaction.

In this Section, we will describe two high-interaction honeypots, both based on virtual machine technology.

Argos

Argos [189] is a containment environment for worms and manual system compromises. It is actually an extended version of the Qemu emulator that tracks whether data coming from the network are used as jump targets, function addresses or instructions. To identify such activities, Argos performs dynamic taint analysis [138] (memory tainting). Memory tainting is the process where unsafe data that reside in the main memory or the registers are tagged. All data coming from the network are marked as tainted because by default they are considered as unsafe. Tainted data are tracked during execution. For example, if we have an `add` operation between a tainted register and an untainted one, the result of the addition will be tainted. Before data enter the Argos emulator, they are recorded in a network trace. As Argos has control of all the operations that happen in the operating system running inside the virtual machine ("guest OS"), it can detect whenever tainted data are tried to be executed or are used as jump targets, e.g. override function pointers. When tainted data are tried to be executed, an alarm is raised and the attack is logged. This log contains information about the attack and specifically registers, physical memory blocks and the network trace. This information is given as input to the signature generation component, which basically correlates information between the memory dump and the network trace using two approaches. The first one locates the longest common sub-sequence between the memory footprint and the network trace. The second one, called CREST, finds the memory location that allowed the attacker to take control of the system. This memory location is found using the physical memory origin and value

of EIP register, that is the instruction pointer register. The value of EIP register is located inside the trace and then trace is extended to the left and right. The extension stops when different bytes are encountered. The resulting byte sequence, along with protocol and port number, is used as a signature. For both approaches, a network trace is useless if data in it are encrypted, for example it is a HTTPS connection. Latest advances of Argos allow it to correlate memory dump with unencrypted data, as Argos comes with modified versions of secure socket libraries for some guest operating systems. The signature generation time is linear to the size of the network trace and generated signatures did not produce false positives for DEFCON and home-grown traces (traces collected at the site of authors).

The basic advantage of Argos is that is able to detect without false positives that an automated attack is taking place, regardless of the application under attack or the attack's level of polymorphism. The major drawback of the Argos approach is the performance overhead. An application running in the Argos environment is 20 to 30 times slower than running in its native environment. A large part of this overhead is due to the underlying Qemu [27] emulation. The rest of the overhead is due to memory tainting and tracking of tainted data. However, as honeypots receive significantly less traffic than production systems, their overhead may be acceptable for some cases.

Minos

Minos[54] is a microarchitecture that implements Biba's low water-mark integrity policy[29] on individual words of data. Minos stops attacks that corrupt control data to hijack program control flow. Control data is any data that is loaded into the program counter on control-flow transfer, or any data used to calculate such data. The basic idea of Minos is to track the integrity of all data and protect control flow by checking this integrity when a program uses the data for control transfer.

Minos requires only a modicum of changes to the architecture, very few changes to the operating system, no binary rewriting, and no need to specify or mine policies for individual programs. In Minos, every 32-bit word of memory is augmented with a single integrity bit at the physical memory level and the same for the general-purpose registers. This integrity bit is set by the kernel when the kernel writes data into a user process memory space. The integrity is set to either "low" or "high" based upon the trust the kernel has for the data being used as control data. Biba's low water-mark integrity policy is applied by the hardware as the process moves data and uses it for operations. If data with "low" integrity are going to be executed, then an attack is taking place.

Minos was emulated on Bochs Pentium emulator. The software Minos emulator is able to execute 10 million instructions per second on a Pentium 4 running at 2.8GHz. The emulated Minos architecture runs for nearly two years without any false positives. Furthermore, various exploits have been launched against Minos to check his detection capabilities. These attacks tried to exploit various types of vulnerabilities such as format string, heap globbing, buffer overflow, integer overflow, heap corruption and double free()'s. All attacks were caught by Minos.

Practical Taint-Based Protection using Demand Emulation

The concept of building a honeypot that is robust against being compromised using memory tainting is also presented in [89]. Unlike other tainting-based approaches, like Argos, that run exclusively inside an emulated environment,the technique of this work switches between virtualized and emulated execution. The reason for this switching is primarily performance. The proposed approach is based on the Xen [25] virtual machine monitor that runs a protected operating system within a virtual machine. Xen provides a virtualized environment on top of which operating systems can

run with very high performance. The containment of the approach is done as follows. When the processor accesses tainted data, Xen switches the virtual machine from the virtual CPU to an emulated processor. The emulated processor runs as user-space application in a separate control virtual machine, referred as ControlVM. The emulator tracks the propagation of the tainted data throughout the system and when no tainted data are accessed, Xen switches the virtual machine back to virtualized execution. The tracking of tainted data is extended to handle disk operations and not only memory and registers. As the proposed approach runs in an emulated environment only when needed, the performance of the approach makes it very attractive. While a fully emulated system that performs tainting runs 88 to 170 times slower, the proposed approach runs 1.2 to 3.7 times slower. The performance benchmark was performed using the LMbench suite [123], which measures time for well-known system calls.

Honeynet.BR

The Honeynet.BR[6] is a member of the Honeynet Research Alliance since June, 2002. The Honeynet.BR team has been involved in the maintenance and expansion of the Brazilian Honeypots Alliance, a network comprised of distributed low-interaction honeypots based on Honeyd, deployed in academic, commercial, government and military institutions. The objective of this project is to increase the capacity of incident detection, event correlation and trend analysis in the Brazilian Internet space. Currently, the alliance enumerates 36 partners. Statistics for traffic collected by members of the alliances are available through the website of Honeynet.BR. These statistics are exported in a daily basis and and include network flow data directed to honeypots of the Brazilian Honeypots Alliance as well as port summary statistics for TCP/UDP traffic data directed to the honeypots.

New Zealand Honeynet project

The New Zealand Honeynet Project[9] is an active meber of the Honeynet Research Alliance. Its research initiatives includes the Client Honeynet Project, the Global Distributed Honeynet project and collection of darknet data with several network telescopes. Concerning the Client Honeyney project, the New Zealand team has implemented several well-known tools like HoneyC (low-interaction client honeypot), Capture-HPC (high-interaction client honeypot) and the Capture BAT analysis tool. The Global Distributed Honeynet (GDH) project is an attempt to develop, deploy, operate and analyse data from a global network of honeynets. It is similar in concept to the Leurre.com or Brazilian Distributed Honeynet, but instead makes use of high interaction research honeynets. The New Zealand Honeynet Project contributes the collected darknet data to REN-ISAC, the Research and Education Networking Information Sharing Analysis Center. Aggregated views of collected data are publicy available[11]

Philippine Honeynet Project

The Philippine Honeynet Project[10] is a non-profit, all volunteer group dedicated to honeynet and I.T. security research and a member of the Honeynet Reseach Alliance. The project runs the Honeynet Activity Monitor which gathers and collates daily data from the Philippine Honeynet. Data labels are based on Snort IDS signatures. Honeynet Activity Monitor currently includes general security activity time and frequency distribution, general security activity totals and percentages, attacks, scans and probe time and frequency distribution, honeynet threat Levels, priority 1/2/3 alert time and frequency distribution, scans and probes totals and percentages protocol distribution,

TCP/UDP/ICMP ports totals and percentages, geographic distribution of attacks and individual attacker time and frequency distribution.

2.1.4 Client-side honeypots

Recent years have seen a large increase in vulnerabilities and exploitation attempts against client applications, such as browsers, e-mail clients, office applications. The WMF and JPEG vulnerabilities for instance ([125, 124]) have shown how the Internet Explorer browser can be compromised and execute arbitrary code to the victim's side. The ineffectiveness of our current detection methods against these exploits is astounding: according to [208], only 1.5% of IDS signatures are based on client-side attacks, although the number of client-based vulnerabilities increases over time.

Traditionally, honeypots have been used as an effective means of discovering novel threats on the Internet. These honeypots have been server side, passively waiting for attacks against them to occur. The shift in the threat landscape to client side attacks has however meant that these kind of honeypots have been rendered blind to many new threats. Detection of these new threats is possible however by applying the honeypot concept to client side attacks. Passive client honeypots can exist as well: E-mail client honeypots are an example of passive client honeypots - they have to wait for an e-mail to arrive (the trigger) in order to detect a threat. Shelia [163] is an example of an e-mail client honeypot that examines e-mail attachments.

Instead of waiting passively for the attackers to contact them, as we have seen so far, many client-side honeypots try to spot locations where malicious content is hosted. Client-side honeypots crawl a communication channel, for example the World Wide Web, searching for malware (they are sometimes deemed as "honey-crawlers"). Although they are not passive systems, they are still characterized as honeypots as they are non-production systems. This is often the case for web browsers. Browsers are attractive targets - they can be found on virtually any machine, deep inside firewall protected networks. E-mails can be used to pass URLs through antivirus filters (an action easier to detect and block if a binary was transmitted instead of a URL) in the hope that a user will click on a malicious URL and infect his machine. Most client honeypot implementations are concerned with browser security.

In this Section strider honeymonkeys and honeyclient, the most popular client-side honeypots, are presented.

Strider HoneyMonkeys

Strider Honeymonkeys system[212] uses *monkey* programs that attempt to mimic human web browsing. A monkey program runs within virtual machines with OS's of various patch levels. The exploit detection system of honeymonkeys has three stages. In the first stage, each honeymonkey visits N URLs simultaneously within an unpatched virtual machine. If an exploit is detected then the one-URL-per-VM mode is enabled. In that mode, each one of the N suspects runs in a separate virtual machine in order to determine which ones are exploit URLs. In Stage 2, HoneyMonkeys scan detected exploit-URLs and perform recursive redirection analysis to identify all web pages involved in exploit activities and to determine their relationships. In Stage 3, HoneyMonkeys continuously scan Stage-2 detected exploit-URLs using (nearly) fully patched VMs in order to detect attacks exploiting the latest vulnerabilities. The exploit detection is based on a non-signature approach. Each URL is visited in a separate browser instance. Then, honeymonkey waits for a few minutes to allow downloading of any code. Since the honeymonkey is not instructed to click on any dialog box to permit software installation, any executable files or registry entries created outside the browser sandbox indicate an exploit. This approach allows the detection of 0-day exploits. During the

first month of honeymonkey deployment, 752 unique URLs were hosting malicious pages, while a malicious web-site was performing zero-day exploits.

HoneyClient

HoneyClient[195] is a honeypot system designed to proactively detect client-side exploits. It runs on Windows platforms and drives Internet Explorer through two Perl scripts. The first script acts as a proxy, while the seconds performs integrity checking. Internet Explorer is set to have as proxy the script. After each crawl is finished, files and registry are checked. In case a change is detected, file and registry key value changes as well registry key additions or/and deletions are logged. The idea of HoneyClient is not restricted to Web crawling. It can be extended to other protocols, like P2P, IRC and instant messaging clients.

Capture-HPC

Capture-HPC [190] is a high interaction client honeypot solution in Java and C++ designed by Christian Seifert of the Honeynet Project. Capture-HPC detects attacks against HTTP aware applications (such as Internet Explorer, Firefox, Safari) by driving them automatically to interact with potentially malicious Web servers through a dedicated virtual machine and observing its system for unauthorized state changes and network activity. The Capture framework consists of a server that can communicate either remotely or locally with multiple clients that reside on the monitored virtual machine. The client process observes registry, file and process changes on the system through modified system calls at the kernel level. Apart from being able to detect the fact that the system has been compromised it is capable of extracting the malware that has been installed.

From the WOMBAT perspective, Capture-HPC is a useful tool for collecting malware that is spread by browser attacks. Additionally, it can detect new (0day) exploits against the latest patched operating system and browser versions.

2.1.5 Wireless Honeypots

One the main purpose of wireless honeypots is to collect statistics and client behavior on an "Open" (unrestricted) wireless access. This access is usually based on IEEE 802.11 [7] but other possibilities are also applicable (e.g. Bluetooth [1]). Of course, this is not limited to this particular aspect, other possibilities exist, like observing the current state of the art on wireless layer 2 attacks. This Section is an overview of articles, research papers or tools that are related to wireless honeypots.

Wi-Fi Honeypots a New Hacker Trap

One of the first article on this area of research [158] was related to a wireless honeypot deployed by the Science Applications International Corporation (SAIC). The architecture was composed of several Open access points and a wired network with several vulnerable computers. The location of this wireless honeypot was of course unknown and not advertized, in order to catch only non-authorized connections. The goal of this experiment was to evaluate if wardriving and hacking into wireless open networks is a myth or not.

Wireless Honeypot Countermeasures

This article by Laurent Oudot [142] gives recommendations in order to design and implement a wireless honeypot. It also describes some requirements to fool wireless hackers, like simulating

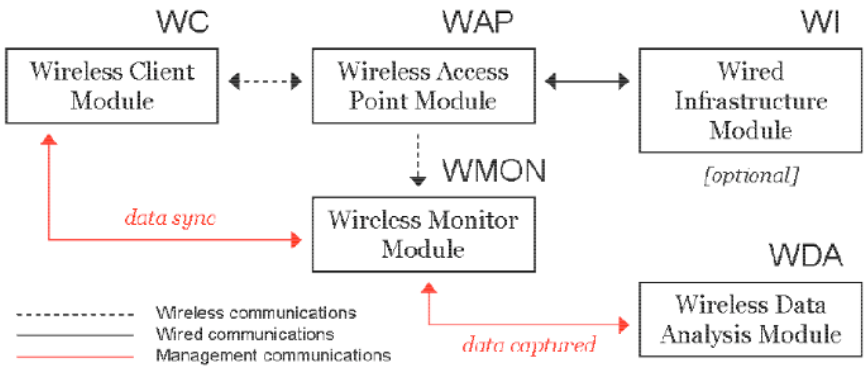

Figure 2.3: Architecture of HoneySpot (source: Spanish Honeynet Project

wireless activity and so on... This paper is a good introduction of the technical requirements when designing a wireless honeypot.

HoneySpot: The Wireless Honeypot

HoneySpot is an architecture designed by the Spanish Honeynet Projet (SHP) that aims at monitoring the attacker's activities in wireless networks. The paper describes several approaches when designing a wireless honeypot. Basically, the design depends on the goals of these kind of experiments. Is it for monitoring dedicated wireless attacks, or, is it for monitoring attacks where wireless technologies are used as a transport protocol for these attacks? This has tremendous impacts on the architecture design. The proposed architecture is composed of several modules. The Wireless Access Point module that will provide the attacker with wireless connectivity. The Wireless Client module that will simulate client activity on the wireless honeypot fooling the wireless hacker and giving enough information in order to launch wireless attacks (such as guessing a WEP key). The Wireless Monitor module that will collect wireless trafic listening for wireless attacks (like a wireless intrusion detection system). The Wireless Data Analysis module that will take care of all reported data by the Wireless Monitor module. Finally, the Wired Infrastructure module that will simulate a real-world wired network with Internet access or emulated network and services. This wireless honeypot architecture is by far the most advanced in terms of features.

HoneyD Configuration Sample

The HoneyD project provides a configuration sample [160] emulating a network architecture. This can be easily used on an HoneyD system where the wireless access is open. This sample configuration can be a good starting point when implementing a basic wireless honeypot.

An investigation into the unauthorised use of 802.11 wireless local area networks

This report [8] presents statistics of three wireless honeypots deployed by the University of South Australia. It exposed that all wireless honeypots observed unauthorized connections, but most were likely due to misconfigurations of automatic association. Some other connections were malicious because compromission attempts were observed on all honeypots. The deployed architecture was based on honeyd and an open wireless access. This report confirms that wireless honeypots can give valuable information regarding attacks statistics on wireless networks.

The Hive

The hive [187] is a pre-configured Debian Live-CD with a wireless honeypot. It is composed of Honeyd and a set of Honeyd scripts for emulating the wired network.

Other Work

Other research topics may be technically similar to wireless honeypots. Karma [223] and HotSpotter [128] are proof-of-concept tools aiming at catching wireless clients that seek for open access points. Whenever a wireless client is catched by this technique, it may be possible to exploit client-side vulnerabilities. Attacking wireless automatic association is considered as a high risk because it is both stealthy, reliable and efficient.

2.1.6 Other research on honeypot systems

This Section is an overview of research papers that are based on honeypot technologies and infrastructures.

ScriptGen

By default, service emulation in honeyd is done by shell scripts, written usually in Perl, that ship with default distribution. The original purpose of honeyd is not to provide accurate scripts (in fact its scrips are just a proof of concept) but rather the framework to hook advanced scripts. Towards this direction, ScriptGen[109] is an effort to build an automated script generation tool. The input for this tool is a tcpdump trace, from which sequence of messages is extracted. A message is defined as the application-level content of a packet. Authors focus on TCP-based protocols only. These messages are used to build a state machine. As the size of the state machine can become very large, as a next step this state machine is simplified using the PI algorithm, an algorithm introduced in the Protocol Informatics Project[26], and an algorithm proposed by authors, the Region Analysis algorithm. Finally, from the simplified state machine a honeyd script is generated. The PI is an algorithm that performs multiple alignments on a set of protocol samples and is able to identify the major classes of messages. The result of the PI alignment is given as input to the Region Analysis algorithm. The aligned sequences produced by PI are examined in a byte per byte basis and for each byte its type (ASCII or binary), value, variability of its values and the presence of gaps in that byte are computed. A region is then defined as a sequence of bytes that have the same type, have similar variability and have, or not, gaps. The quality of the results of ScriptGen is limited by the number of samples we use in the state machine generation. For example, if we use two samples, "GET /file1.html" and "GET /file2.html", ScriptGen would generate a state machine that is triggered by the "GET /file?.html" regular expression. This is wrong as any other request will not be handled by the generated state machine.

A Pointillist Approach for Comparing Honeypots

Pouget and Holz in [157] used the honeypot architecture of Leurre.com platforms to compare low- and high-interaction systems. The low-interaction honeypot setup is the same as this of a Leurre.com platform. The high-interaction honeypot setup was the same as the low-interaction one but instead of using modified version of honeyd, a VMware client was used. All virtual guests had adjacent IP addresses and they were monitored for 10 continuous weeks (August to October 2004). Both setups observed around seven thousands attacks. Attacks were classified into three types: first type (Type

I) contains attacks that target a single host, Type II includes attacks that target 2 hosts and finally Type III includes the attacks that targeted all three hosts of the setup. Most of the attacks (around 67%) are Type I. However, for these attacks, high-interaction setup received 40 times more packets and this is due to the fact that they target talkative services. As low-interaction honeypots do not emulate service in depth, they receive considerably less packets. Around 4% of the attacks, targeted only 2 out of 3 systems of each setup. However, the majority of these attacks are incomplete Type III attacks. Concerning Type III attacks, an interesting fact is that all IP sources were observed on both environments. Authors reached three main conclusions. First, it is not necessary to deploy honeypots using hundreds of public IP addresses in order to identify scan activities against large block IPs. Second, low-interaction honeypots bring as much information as high interaction ones when it comes down to global statistics on the attacks. Finally, both interaction levels are required to build an efficient network of distributed honeypots, using high-interaction as a means to configure low-interaction ones.

Configuration of Dark Addresses

Honeypot networks are usually setup based on manual and ad-hoc configurations. However, such configurations do not lead to good visibility and are vulnerable to discovery. A typical example is when the production network consists of Linux workstations whose SSH service is attacked, while honeypots of the same domain emulate services like Microsoft SQL server. Taking into account the huge diversity in network and host configurations and the fact that vulnerabilities increase year by year, the task of configuring honeypots becomes even harder. Sinha et al. in [179] address this problem and propose an automated technique to generate honeynet configurations. Authors examined four configurations. First one was the one used by the Internet Motion Sensor project, where incoming connections are accepted but no application-level response is provided (All TCP Responder). The second one was the default configuration of Honeyd (Generic Honeyd) and the third one was random combinations of chosen OSes and services created by RandomNet[181] (Random Honeyd). The last configuration was the default one of Nepenthes system. The configurations were compared against the top 10 open TCP ports of an academic /16 network. All configurations failed to capture attacks on several ports that were attacked on the production environment and at the same time made certain ports, like TCP port 139 (Microsoft RPC service), appear more important than others. Such behavior does not only affect the visibility of honeypots but also allows attacker to identify them.

Authors used a simple methodology to show that the configurations mentioned before expose honeypots to detection. They constructed a set of tests to make profiles for each host. These tests included checking ambiguities in implementation of TCP software (*tcp* test), identification of a web server and its configuration (*http* test) and finally a scan to TCP ports 1 to 1024 to discover open ones (*ports* test). Hosts are grouped by test values and then subnet is compared against the network as a whole for the values of the test. Then values of the test are aggregated and values of next test are compared. This process iterates over the determined test order until the last test. Tests are ordered by increasing entropy. To identify anomalies in a test, authors use z-statistics. The subnets with the highest anomaly score ideally should be the honeynets. Their methodology was evaluated by embedding different honeynet configurations into six networks, two of which were academic network and the rest four were web-server farms. The honeynet configurations were created using all four approaches (generic honeyd, random honeyd, all TCP responder and Nepenthes). Results yielded that is easy to realize honeynets created by all four configurations, although someone would expect that configurations like Generic Honeyd and Nepenthes would be more resilient to identification.

The automated creation of configurations is a challenging task due to two main reasons. The

first one is that there is variation in the threat landscape. Ports that are frequently attacked in one month, maybe rarely contacted the next month and vice versa. Authors confirm this variation by observing data collected over a 5-month period. The second one is the heterogeneity in operating systems and services. By examining 5512 hosts, 352 unique TCP stack implementation were found, 1237 hosts were running 241 different version of web servers. The proposed technique is based on profiling the existing network and generating configuration according to the extracted profiles. The profiles proportionally represent the network and are individually consistent. Individually consistency guarantees that a honeypot will not have an unrealistic profile and that it will match a live host. To achieve proportionality, real hosts are sampled using stratified sampling. Stratified sampling involves separating the population and allocating the size to be sampled proportionally at each point in the hierarchy. Additionally, weights in the values of some tests are supported. That means that if a port is preferred open than closed, a host with that port open will replace a host with that port closed during the profile extraction. Honeynets with configurations created by that process were embedded in the same five networks mentioned before. Results show that honeynets provide better visibility into vulnerable population. The distribution of services and operating systems is close to the vulnerable population. Furthermore, honeynet configurations are more resistant to fingerprinting when configured using authors' approach.

Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic

M. Bailey et al. in [22] examine the properties of darknets in order to determine the effectiveness of building hybrid honeypot systems, e.g. systems that combine monitoring of unused address space with honeypot farms. Their goal is to analyze darknet traffic and filter out redundant traffic that will impose unnecessary overhead to honeypot farms. They try to identify threat characteristics that will enable them to limit the number of connections that reach honeypots. Their measurements and analysis was done over 60 darknets in 30 organizations, a monitored space of 17 million addresses. The dark address space was monitored using the Internet Motion Sensor architecture presented in Section 3.3.10. The 14 IMS sensors monitored networks with variable size, ranging from /25 up to /17, over a period of 10 days (mid-August 2004). Initially, the source IP addresses were evaluated. The observed distribution showed that 90% of the packets were sent from less than 10% of source IP addresses. Secondly, the destination port distribution was examined. 90% of the packets target less than 1% of the destination ports. However, as the cross product of unique source IP addresses and total destination port is large, the top 10% source IP addresses were evaluated separately in terms of destination port they contact and unique payloads they send. Over 55% of these IP addresses contacted a single destination port, 90% contacted less than six while 99% contacted less than ten. In terms of the paylod they send, 30% of addresses send only one payload, 70% send two or less and 92% of addresses send less than ten payloads.

Authors evaluated three methods of source-based filtering previously reported in [144]: source-connection, source-port and source-payload. In source-connection filtering, N connections from a single source are recorded, and all subsequent traffic from that source is ignored. In source-port filtering, N connections are maintained for every source and destination port pair. In source-payload filtering, the first N payloads sent by a source are used to define all future behavior by that source. The average reduction due to these three methods for N=1 was ranging from 86% up to 97%, but there was huge variations over time. In some time periods the reduction was dropped down to 47% and the main reason behind that behavior is that there is little overlap across subnets. Any additional darknet block added for monitoring will bring its own sensor-specific events. Based on that two observations, authors constructed a new filtering mechanism. Author's approach examine the distribution of unique IP addresses to a specific port.

Every hour, each darknet is queried for the number of unique source IP addresses that have contacted it with destination port x. For each hour, the number of unique source IP addresses contacting destination port x at each darknet is added up. This data is scanned over, one hour at a time, comparing the average (per hour) over the event window (last event window hours) to the average over the history window (last event window product the history factor) hours. If the ratio of event window average to history average is greater than the event threshold, an event is generated. These events are then filtered based on whether they are global or local, via the coverage threshold. The coverage threshold defines the number of darknets that would have generated an event individually for a threat. Their alert mechanism was able to identify such as a MYSQL worm, WINS and Veritas Backup Agent vulnerabilities, along with most major events that took place during the 4-month deployment phase of the technique.

Detectability of honeypots

Detectability of honeypots is an interesting research area. Nevertheless, we can only find a limited number of works in this area. In the article published in [52], three techniques were introduced to detect three subsystems of honeypots. The first one detects Sebek, the kernel module used in honeynets for logging host activities. Sebek hides itself from the kernel module list and this is done by a module called *cleaner*. After cleaner runs, the lsmod command does not provide any visual output that a module is running. However, information still remains in /dev/kmem and a simple grep of "insmod_sebek_S.data" reveals the existence of Sebek. The second technique introduces a way to detect honeyd. The idea is based on the fact that by default honeyd resets the connections it cannot handle. Even when the SYN packet is malformed, the honeyd will respond with reset. A random scanning tool was used to identify honeyd systems by looking at the resets at random ports. The last technique identifies if an operating system runs inside VMware. As VMware is commonly used in the setup of honeypots, malware may use the technique to detect it runs on a honeypot or to prevent itself from being executed inside a virtual guest. The idea is very simple; the device names of virtual guest OS reveal the existence of VMware : "Model Number: VMware Virtual IDE CDROM Drive". Although this can be fixed, scanning of the devices reveals even more information as VMware does not return all necessary information.

Detection techniques against Honeynets, and especially Sebek, are also described in [69]. The first technique checks if network is congested due to Sebek. As Sebek client sends a packet to Sebek server for each read() operation, if one can call read() thousands times per second, this leads to network congestion. The detection is as follows. A local host is ping()'ed. Local host can be reached within a millisecond and this time is recorded. After the first ping, a second one is sent but now a *dd* operation runs in the background. According to measurements, the response time changed from 0.7 milliseconds to nearly 5 seconds. The second proposed approach is to check the system call table and observe the difference between the addresses of sys_read and sys_write. As sys_write is successor of sys_read in the system call table, these two functions originally have a few kilobytes distance. When Sebek overwrites the sys_read pointer, this distance increases to tens of megabytes. Authors also propose a technique to detect the presence of honeywall. As honeywall by default limits the number of connections to 15 per day, it is trivial to detect if the 16th connection is dropped or not, provided that connections target a known destination host. Additionally, an attacker can send packets that match snort inline signature (snort inline runs in the honeywall) to a machine it controls and then check if this packet was rewritten according to the matching signature.

Monitoring high interaction honeypots

Network monitoring A possible way of monitoring the status of a high interaction honeypot consists in monitoring the network traffic that it generates.

Honeynet project's Honeywall [4] is normally setup as a transparent bridge between the farm of high interaction honeypots and the Internet. The containment techniques employed by Honeywall combine together two different techniques.

Firstly, honeywall takes advantage of a set of iptables rules to limit the ability of the attacker to take advantage of the bandwidth available to the honeypot. The default rules are extremely strict, and allow the honeyot to generate only 20 TCP connections, 20 UDP packets, 50 ICMP packets per hour. This prevents the attacker from being able to run DoS attacks against other victims.

While these rules are effective in preventing this kind of attacks, a single TCP connection can be enough for an attacker to compromise another victim using the honeypot as a stepping stone. In order to address this class of attacks, Honeywall takes advantage of Snort-inline. Snort-inline is a modification of the popular IDS Snort [164] to allow the modification of packets that match a given set of rules. Honeywall uses this tool to perform a *sanitization* of the exploits: the network bytes known to be essential for the exploit to succeed are replaced with innocuous ones. The purpose of the sanitization technique is to prevent the attacker from succeeding in the exploit, and at the same time invite him to try again, eventually with different exploits or different exploit configurations.

While a solution such as Honeywall can be very effective in most cases, it is important to understand that it is a *knowledge-based* solution. The Snort-inline signatures used by Honeywall are able to sanitize only those exploits that follow known attack patterns. It is still theoretically possible for an attacker having a specific interest in taking advantage of the honeypot to study these signatures and find a way to evade them.

An interesting network monitoring technique was developed by Georgia Tech, the BotHunter [86] tool. BotHunter is built upon Snort and specifically aims at the detection and the collection of data on infections of self-propagating malware. Its nature is thus comparable to a special-purpose IDS, but the authors show in [86] how BotHunter can be coupled with a honeynet and be used as a data collection and containment mechanism. BotHunter analysis is based on a high level model of a self-propagating malware, with special interest to IRC-based bots. BotHunter generically models the malware infection as composed of 5 different stages:

- E1: External to Internal Inbound Scan
- E2: External to Internal Inbound Exploit
- E3: Internal to External Binary Acquisition
- E4: Internal to External C&C Communication
- E5: Internal to External Outbound Infection Scanning

BotHunter tries to independently detect each of these 5 stages, and takes advantage of correlation techniques to infer the presence of a malware infection. In case an infection is detected, BotHunter is able to generate detailed reports on the infection process, along the 5 detection stages, such as information on the exploited TCP port, the malware binary being pushed to the victim, and information on the eventual Command & Conquer channel. BotHunter does not prevent malware from trying to propagate to other hosts: differently from Honeywall, outbound infection scanning is not blocked. Being the detection focused on a behavioral model rather than exploit-specific signatures, BotHunter can be used to employ more effective containment techniques. In [86] the

authors react to a detected infection tainting the corresponding high interaction honeypot and then sanitizing it.

Host monitoring High interaction honeypots, with their higher level of interactivity, provide rich information on the attacker activity. This level interaction translates into verbose network information, but also in detailed information on the modifications performed by the attacker at *host level*. This information can be exploited to have exact information on the success of attacks and on the status of the honeypot to implement a very reliable containment.

A more sophisticated system was proposed in [222]. The Honeybow high interaction honeypot system combines together three different monitoring techniques to minimize the ability of an attacker to escape from detection. These techniques focus on a specific aspect linked with network exploits: the generic need for an attacker to push an executable file to the victim taking advantage of a successful exploit.

- **MwWatcher.** Similarly to Sebek, MwWatcher runs within the virtualized high-interaction honeypot and logs suspicious modifications to the filesystem.
- **MwFetcher.** MwFetcher monitors the virtualized host from outside, accessing the virtual drive of the honeypot and detecting changes to the filesystem.
- **MwHunter.** MwHunter analyzes the network stream taking advantage of Snort-inline, and detects the presence of (unencrypted) executable samples in the network stream.

greg

The authors claim that the combination of these three techniques maximizes the probability of correctly detecting an infection. Whenever a detection is detected, the honeypot is automatically sanitized and restored to a clean snapshot.

2.2 Malware collection systems

2.3 Attack detectors

Attack methods such as buffer overflows on heap or stack, double frees and format string exploits, have been the subject of research by the security community for years. In this section, we review the state of the art in attack detection methods as applied within the context of honeypot systems.

Many existing automated detection systems tend to incur a fairly large ratio of false positives in attack detection and use of signatures (e.g., VirusThrottle [217], EarlyBird [178], AuthoGraph [91], HoneyStat [61], and HoneyComb [101]. Although these systems may play an important role in intrusion detection systems (IDS), they are not suitable for fully automated response systems.

Both Stackguard, Stackshield and gcc extensions have been used to protect against stack smashing attacks [36, 94]. Later research has suggested that many of these methods can be easily bypassed [34]. There are patches for most OSs to make the stack non-executable, but this introduces other problems: for instance, trampolines [2] rely on stacks being executable, and (at least in Linux), so do signals. However, such mechanisms can sometimes be bypassed. Buffer overflow detection and protection methods exist in abundance [175, 35, 132, 165]. They make it difficult/impossible to overwrite specific addresses so as to divert the control flow, e.g., by modifying the way in which code and data are stored in memory.

Some existing format string protection methods afford safety by way of a patch to `glibc` which counts arguments to `printf()` [37] and by using type qualifiers [198]. Both approaches require recompilation of the code. Code injection problems have been addressed by instruction set randomisation and detection of attempts to perform syscalls from illegitimate addresses [71, 80, 93]. To generate signatures and/or perform detailed analysis, instruction set randomisation is not very useful. The syscall protection offered by Dome [93] is potentially interesting. However, its scope is limited (to syscalls) and it incurs inconvenience (static analysis of every application is required to determine legitimate addresses for performing syscalls).

A different approach is to guard against overflows and attacks in hardware. For instance, StackGhost [118] protects the stack on Sparc architectures. Similarly, [79] uses dynamic information flow analysis to guard against overflows and some format string attacks. None of these mechanisms are widely available for the most commonly used processor/OS combinations and indeed, to the best of our knowledge [79] has not progressed beyond simulation. Instead of real machines Dunlap and Garfinkel suggest virtual machines [81, 82]. Similar work is presented in [200] which uses a modified version of the Dynamo dynamic optimiser.

As mentioned earlier, taint analysis is used by many projects such as TaintCheck [139], Minos [55], Vigilante [117], and Argos [151]. Most of the existing work assumes deployment on dedicated honeypots. This is mainly due to performance reasons. Likewise, client-side honeypots tend to be dedicated machines also [213, 129]. As a result, these techniques suffer from several problems:

1. Honeypot avoidance: an attacker may create a hitlist containing all hosts that are not honeypots and attack only those machines.

2. Configuration divergence: the configuration of honeypots often does not match exactly the configuration of production machines. For instance, users may have installed different versions of the software, plugins, or additional programs. Honeypots only reflect a limited set of configurations. Indeed, high interaction honeypots typically have a single configuration.

3. Management overhead: honeypots require administrators to manage at least two installations: one for the real systems, and one for the honeypot.

4. Limited coverage: even if a honeypot covers a sizable number of IP addresses, it may take a while before it gets infected. This is especially true if the honeypot only covers dark IP space. Moreover, the address space that is covered is limited by the amount of traffic that can be handled by a single honeypot.

5. Server-side protection: most honeypots mimic servers, by sitting in dark IP space and waiting for attackers to contact them. Unfortunately, the trend is for attackers to move away from the servers in favour of client-side attacks [72, 167].

A interesting exception includes the work on speeding up taint analysis by switching between a fast VM and a heavily instrumented emulator by Ho et al. discussed earlier [14]. One drawback of the method (besides an overhead that is still considerable compared to native execution) is that it can only be installed by, say, home users willing to completely reconfigure their systems to run on a hypervisor.

Another way to deal with performance penalties is by running in slow mode 'on demand'. In essence, one slices up program execution in the temporal domain or spatial domain.

An example of slicing in the temporal domain is found in Eudaemon [150]. Eudaemon is a technique that aims to blur the borders between protected and unprotected applications, and brings

together honeypot technology and end-user intrusion detection and prevention. It is able to attach to any running process, and redirect execution to a user-space emulator that will dynamically instrument the binary by means of taint analysis. Any attempts to subvert control flow, or to inject malicious code will be detected and averted. Eudaemon can move an application between protected and native mode at will, e.g., when spare cycles are available, when a system policy ordains it, or when it is explicitly requested. The transition is performed transparently and in very little time, thus incurring minimal disturbance to an actively used system.

A different way of slicing is known as application communities [114]: assuming a software monoculture, each node running a particular application executes part of it in emulated mode. In other words, applications are sliced up in the spatial domain and a distributed detector is needed to cover the full application. While it is true that the OS used by communities tends to be uniform, but variation tends to exist in applications, due to plug-ins, customisations and other extensions.

In a later paper, the same groups employs selective emulation to provide self-healing software by means of error virtualisation [115]. Again slicing is mostly in the spatial domain. As far as we are aware, neither of these projects supports taint analysis. Indeed, it seems that for meaningful taint analysis, the tainted data must be tracked through all functions and, thus, *selective* emulation may be more problematic.

Another interesting way of coping with the slowdown (and indeed, a way of slicing in the temporal domain for servers) is known as shadow honeypots [18]. A fast method is used to crudely classify certain traffic as suspect with few false negative but some false positives. Such traffic is then handled by a slow honeypot. Tuning the classifier is delicate as false positives may overload the server.

Rather than incurring a slow-down at the end users' machine, many projects have investigated means of protection for *services* running on user machines by way of signatures and filters. Such projects include advanced signature generators (e.g., Vigilante, VSEF, and Prospector [117, 31, 180]), firewalls [42], and intrusion prevention systems on the network card [65].

For instance, Brumley et al. propose vulnerability-based signatures [31] that match a set of inputs (strings) satisfying a vulnerability condition (a specification of a particular type of program bug) in the program. When furnished with the program, the exploit string, a vulnerability condition, and the execution trace, the analysis creates the vulnerability signature for different representations, Turing machine, symbolic constraint, and regular expression signatures.

Packet Vaccine [210] detects anomalous payloads, e.g., a byte sequence resembling a jump address, and randomises it. Thus, exploits trigger an exception in a vulnerable program. Next, it finds out information about the attack (e.g., the corrupted pointer and its location in the packet), and generates a signature, which for instance can be based on determination of the roles played by individual bytes. To determine this, Packet Vaccine scrambles individual bytes in an effort to identify the essential inputs.

Vigilante relies on the possibility to replay attacks in order to generate Self-Certifying Alerts (SCAs), an extremely powerful concept that allows the recipient of an alert to check whether the service is indeed at risk [117]. If so, it may automatically generate a signature. The false positives rate seems to be zero.

As mentioned earlier, the signature generators for the above projects may be capable of handling zero-day attacks, but they produce them by means of dedicated server-based honeypots. Hence, they do not cater to zero-day attacks on client applications. To some extent the problem of zero-day attacks also holds for virus scanners [188], except that modern virus scanners do emulate some of the data (e.g., some email attachments) received from the network. However, remote exploits are typically beyond their capabilities.

Many groups have tried to use such fast detection methods like address space randomisation

and perform more detailed instrumentation on a different host by replaying the attack [197]. In our opinion, replaying is still difficult due to challenge/response authentication, although promising results have been attained [58, 136]. More importantly, the heavily instrumented machines that perform the analysis may become a bottleneck if many attacks are detected. It inherently scales because it employs the users' machines.

Polymorphic attacks tend to much harder to detect than monomorphic attacks. For instance, one can no longer look for simple byte patterns using Snort. Previous work on detection of polymorphic attacks focused on techniques that look for executable code in messages, including: (a) abstract or actual execution of network data in an attempt to determine the maximum executable length of the payload [196], (b) static analysis to detect exploit code [43], (c) sled detection [16], and (d) structural analysis of binaries to find similarities between worm instances [106].

Transport-layer filters independent of exploit code are proposed in Shield [206] with signatures in the form of partial state machines modeling the vulnerability. Specific protection against instruction and register shuffling, as well as against garbage insertion is offered by semantics-aware detection [45].

A related project, PolyGraph [137], fingerprints attacks by looking at invariant substrings present in different instances of suspicious traffic. The idea is to use these substrings as a signature. Such methods are vulnerable to the injection of noise in the data stream [147].

Various groups have proposed anomaly detection for catching polymorphic attacks. PAYL [209] builds a model of the byte distribution of normal traffic and compares real traffic with this model. Increasingly sophisticated mimicry attacks [102, 84] are a problem and spark many new developments in this direction [76, 74, 105]. SigFree [211] observes that overflow attacks typically contain executables whereas legitimate requests never contain executables, and blocks attacks by detecting the presence of code.

A final approach to deal with polymorphic buffer-overflow attacks on heap or stack is what is known as Prospector [180]. Heap and stack buffer overflows are still among the most common attack vectors in intrusion attempts. Prospector uses a variant of dynamic taint analysis to identify the information that needs to be tracked to pinpoint the bytes in an overflow attack. Using these bytes, it generates a signature based on the length of protocol fields that is robust against polymorphism and both more accurate and reliable than the relatively weak signatures generated by COVERS [112].

3 Existing collection infrastructures

Malware sample collection is the first important step in the process of characterizing and analyzing malicious activities. It is crucial to have a wealth of information in order to achieve high quality results in the subsequent steps, such as characterization mechanisms and threat intelligence.

Also, there are infrastructures maintained by WOMBAT partners that are already constantly collecting malware. The Leurré.com project directed by Eurécom Institute, VirusTotal malware samples, Symantec DeepSight services, honeypot sensors maintained by FORTH and the early warning system operated by CERT Polska can provide enough information to stimulate further research on malware analysis and threat intelligence. Virustotal, operated by Hispasec, offers a free service for scanning suspicious files using several antivirus engines. Companies, institutions, organizations and individuals can submit malware samples that are scanned by the VirusTotal service using 28 different antivirus engines. The number of malware samples received by the service is nearly ten to twelve thousands per day, a quantity that makes Virustotal service an excellent starting point for malware collection.

3.1 Internet telescopes

Internet Telescopes observe a single large portion of the unassigned IP space in order to observe *global* events. Moore et al. showed in [126] how Internet Telescopes can be compared to an astronomical telescope. Increasing the number of IP addresses being monitored (their "resolution"), Internet Telescopes increase their visibility into fainter, smaller, further and older objects. Internet Telescopes have been used to detect global threats such as the Witty worm [174] or the evolution of DoS attacks [127].

Due to the high amount of IPs involved in this kind of deployment, often internet telescopes configure themselves as purely passive sinks for Internet traffic. Examples of such deployments are the CAIDA internet telescope and Team Cymru Darknet Project.

In order to increase the level of interaction of Internet Telescopes and coping with the scalability problems, different filtering solutions have been proposed in the literature. Pang et al. propose in [144] some filtering technique to drop repeated activities, for instance allowing connections from one attacking IP to only N telescope IPs before blocking the activity. These filters allow them to take advantage of stateless responders for common protocols, that continue the interaction until a given activity is *distinguishable* from others.

A more complex solution is GQ [56, 57], that takes advantage of dynamic filters and protocol learning techniques to filter out uninteresting activities and rely the selected activities to farms of fully fledged high interaction honeypots.

An important limitation of the Internet Telescope schema consists in the possible bias derived from the identification of the monitored IP range. Staniford et al. showed in [183] how self-propagating malware could take advantage of a-priori knowledge of the active IPs to avoid these "black holes" in the IP space and consequently avoid detection.

We review more in details the telescopes managed by CAIDA [38], ATLAS [15] and the Japanese NICTer project [141].

3.1.1 CAIDA

The Cooperative Association for Internet Data Analysis (CAIDA) [38] initiative collects and promotes tools and methodologies to analyze and maintain the global infrastructure of the Internet. The data is collected through a `/8` network collecting non-legitimate traffic only and representing no more than 0.4% of the overall address space of the Internet.

Data collected by CAIDA tools includes DNS data, topology traces, RTTs, and routing data. For the purposes of WOMBAT it is interesting to underline that CAIDA also keeps passive information on security events that are widely visible on the global scale such as malwares (mostly Internet worms) and distributed denial of services.

3.1.2 ATLAS

The Active Threat Level Analysis System (ATLAS) [15] collects data from distributed darknets into a centralized database. ATLAS sensors are predominantly deployed into ISPs networks and collected data is stored in the ATLAS data centers; captured data includes honeypot-captured payloads, scan and IDS logs, DoS statistics, news & vulnerability reports, malware samples, phishing infrastructure data, and botnet data.

Instead of using fully-automated procedures, raw data is analyzed by the team of ATLAS security experts and then shared with customers as well as with producers of industry-leading detection systems. As for the end user, ATLAS provides the web interface and, additionally, an enhanced Firefox search plug-in.

Even though ATLAS output knowledge base is potentially more accurate and enriched than the one coming from fully-automated infrastructures, according to the terms of use [135] this initiative is evidently industry-oriented rather than sharing/research-oriented.

3.1.3 NICTER

Network Incident analysis Center for Tactical Emergency Response (NICTER) [141] is composed of four subsystems to analyze the traffic at different levels: *macro* (i.e., distributed sensors to capture and monitor the darknet traffic), *micro* (i.e., used to capture critical payloads such as malwares), *payload* (i.e., an analyzer and a code analyzer to extract malwares' characteristics and behaviors). Results are stored in a database called Malware kNOwleadge Pool (MNOP) and the NemeSys component enchains the phenomena trying to discover root causes. Once it has been given an attacking host observed at *macro* level, the correlation analyzer outputs a list of malwares that have similar network behavior (i.e., scans) as the host.

Finding the root causes of the observed network attacks, NICTER provides a much clearer view of what is happening in the Internet. It also provides an animated graphical representation of the overall activity through a 3D visualization engine.

3.1.4 IUCC/IDC Internet Telescope

The IUCC/IDC Internet Telescope [88] operates at the Israel InterUniversity Computation Center, where a /16 address dark space is monitored to capture back-scatter packets coming from all over the Internet. Like other telescopes, IUCC/IDC classifies the captured packets as "Host/Port scanning", "Backscatter from spoofed DDoS attacks" and "Configuration Mistakes" (e.g., a short-life packet caused by a misconfigured computer connected to the Internet).

3.1.5 Team Cymru

Team Cymru[59] is a specialized Internet security research firm dedicated to collect data and study malicious activities. Team Cymru monitors specific Internet critical infrastructure. Monitoring activities focus on DNS and BGP. The DNS monitoring includes both the DNS service as well as network connectivity to the given name server. The network status of each server's IP, containing prefix and ASN, as well as observed DNS response times from a variety of points within the network are tracked. The BGP monitoring is based on peering with over 100 BGP-speaking routers, providing a granular view of the internal routing tables. BGP statistics that are collected include Internet routing table prefix count, Internet routing table delta, count of prefixes with inconsistent origin ASNs, list of prefixes with inconsistent origin ASNs, Bogon prefixes and the origin ASNs and BGP announcements and withdrawals. ASN analysis and bogus ASN reports are also provided. Team Cymru also monitors darknets, allocated subnets that do not run any active services. For these darknets simple statistics are gathered and more specifically total traffic overview in terms of packets per seconds and bits per second.

3.2 Log sharing and similar initiatives

We have introduced honeypots as the primary way to collect information on malicious activities. Other sources of information exist, such as Intrusion Detection System logs or firewall logs, and can be aggregated to build global pictures of the Internet attacks. Among the main approaches for collection and sharing of logs from heterogeneous sources (e.g., firewalls, IDSs, even from end users) we have reviewed the following. We recall that they are very interesting ([154, 153, 152]) even if most of them provide basic statistics about *global* current threats observed by different sources (i.e. hits per port, hits of specific malware as detected by an IDS, etc.). It must be underlined that, as the data sources are uncontrolled, the resulting datasets could also contain a large fraction of events that do not necessarily represent actual attack traces, hence leading to a high false positive rate if used for production warning systems.

The main problem of this kind of log aggregation approaches is the heterogeneity of the different sources of information. There is a considerable semantic difference between the events recorded by different systems. As shown in [63], the aggregation of logs generated by diverse sources lacks the semantic and the homogeneity necessary to perform correct analyses and fact inferences.

3.2.1 Internet Storm Center and DShield

The *Internet Storm Center* (ISC) is a large-scale initiative by [166]. Beside providing a meter indicating, at a glance, the overall threat level, the aim of this initiative is to provide a worldwide distributed intrusion detection system, DShield [70]. The project is public and any end-user or organization may contribute submiting firewall logs, even anonymously, that are processed and summarized to discover trends or unexpected activities. They also provide an early warning service.

DShield is a very well known source of information about the latest Internet threats. It accepts data from any volunteering contributor, through various scripts that can be installed locally. The scripts parse firewall logs and submit them to a central point in a standard format. Logs are analyzed centrally and trends in destination port activity are discovered. The results are presented publicly in the form of port activity graphs and tables, a map with the most often attacked ports per continent, statistics about most often registered autonomous systems, blacklists of offending netblocks etc. This analysis is not made available in real time.

From the WOMBAT perspective, the main advantage of DShield is the huge amount of sources that contribute security event data. These sources lack homogeneity, making their datasets difficult to compare. Furthermore, they submit information based on packet headers. Such information is useful from an anomaly detection point of view or to judge the scale of the impact of a security incident, making DShield a potential supplier of context information for WOMBAT.

3.2.2 MyNetWatchman

Similarly to the ISC, MyNetWatchman [131] collects data coming from registered users' firewalls. Events are then aggregated and collected into the incident database that can be queried through the web interface on the home page. MyNetWatchman is compatible with most of the firewall logging formats, including home users desktop applications. An interesting feature is that the reports include the status of the notification of the incidents to the ISPs.

Collection is achieved through a client application (agent) that can be installed by a user to automatically forward data in near realtime to a central analysis server. The central analysis server aggregates collected security events and is capable of sending alerts to ISPs responsible for the offending IPs, based on how many different agents see the same source IPs probing sets of ports. A public web page is provided that lists various statistics, including the highest changes in port activity as well as the most often attacked ports. A list of most often offending ISPs is also provided. Contributors to the system have access to personalized pages which offer additional information on reported events, such as their incident handling status, based on responses from the ISPs.

From the WOMBAT perspective, myNetWatchman is a form of Internet anomaly detection system that can alert about shifts in scanning patterns, coming from worms and botnets. No information other than IP, ISP and port activity can be obtained, limiting its potential use to supplying contextual information about malware, which has to be collected elsewhere.

3.2.3 Talisker Computer Network Defense Operational Picture

The Talisker Computer Network Defense Operational Picture [48] was designed to provide near real time information on new and emerging network security threats. Its intended clients are Government and Military networks, but as the service is publicly available it can be used by anyone. Talisker provides high level information from a wide variety of security sources. This includes threat level indexes from other security sites (such as SANS ISC, ATLAS and antivirus vendors), geographical distribution of attack sources, latest detected threats, latest vulnerabilities, security tool versions, intrusion prevention updates and general security news. The focus on aggregating and visualization high level security information makes Talisker different than most systems analyzed in the study, which tend to collect raw data at the packet or operating system level.

From the WOMBAT perspective, the Talisker Computer Network Defense Operational Picture, while interesting is of limited use, as this sort of high level and abstract information cannot be correlated in an automated manner with collected malware, requiring a human operator to identify any interesting activity instead.

3.2.4 Symantec DeepSight Threat Management System

The Symantec DeepSight Threat Management System and Symantec Managed Security Services consists of more than 40,000 sensors monitoring network activity in more than 180 countries and comprehensively tracks attack activity across the entire Internet. From the WOMBAT perspective, this is certainly one of the most reliable and rich source for malware code samples and related

knowledge. In fact, Symantec also retrieves and stores malicious code data along with spyware and adware reports from over 120 million client, server, and gateway systems that have opted into sharing such reports within the agreed upon terms of privacy and anonymization.

Another minor but valuable source of data for WOMBAT is Deepsight Extractor, a free client tool able to parse and submit to a central database logs generated by a number of different network appliances and personal firewalls, accessible through a web interface along with detailed statistics on the submitted logs.

3.2.5 SURFids

This project is founded by SURFnet in the Netherlands and relies on Nepenthes and Argos for detecting attacks. Current Distributed Intrusion Detection Systems (D-IDS) are most often based on a client-server approach where the client is called a sensor. These sensors often contain a honeypot and/or a passive analysis tool like snort. According to SURFnet, this approach has four major disadvantages:

- The sensor must be upgradeable in order to add future honeypots and new signatures.
- The sensor may be vulnerable to the exploits used against the honeypot and passive analysis software.
- The distributed IDS will generate false positive alerts.
- Installing and running the sensor is not plug and play.

In order to avoid these disadvantages SURFnet used a different design for a distributed IDS (D-IDS). This approach is based on the following rules:

- The sensor should run out-of-the-box.
- The sensor should be completely passive and therefore maintenance free.
- The D-IDS should not generate any false positive alerts.
- A sensor should be able to run in a standard LAN.
- Comparison of statistics generated by sensors and groups of sensors should be possible.

In SURFids' approach, an ordinary workstation is used as a sensor. The workstation is turned into a sensor by booting it from an USB stick containing the SURFnet D-IDS sensor software. This USB stick contains a remastered Knoppix distribution and uses OpenVPN to start a layer-2 tunnel to the D-IDS server. The layer-2 tunnel is put in bridging mode with the network interface of the sensor. Next, a DHCP request is made from the D-IDS server through the tunnel into the client LAN. This request allows the D-IDS server to obtain an IP-address on the client LAN and then bind it on a virtual interface containing a honeypot.

Virtually, the D-IDS server will be present on the client LAN and attackers will think they are attacking a host on the client LAN. The honeypots used on the D-IDS server are Nepenthes (for known attacks) and Argos (for zero-day attacks). If an attacker triggers the honeypot it is considered a malicious attack and the honeypot attempts to retrieve the malware that an attacker tries to put on the host, which the attacker thinks is compromised. All attacks are logged into a PostgreSQL database and users are able to view detailed information about the attacks through a web interface.

3.2.6 McAfee SiteAdvisor

SiteAdvisor [121] by McAfee Inc. can test and rate virtually all sites on the Internet. Users can provide direct feedback to other users and to McAfee analysts regarding their personal experience about the sites they visit on the Internet. However, the results from SiteAdvisor should not be taken as definitive nor complete, since site ratings are based on an "advanced patented technology" which is not disclosed. McAfee SiteAdvisor works as a browser extension on both Internet Explorer and Firefox. The last public, yearly report has been published in June 4th, 2008 [122].

3.3 Distributed honeypot architectures

After 2003, the proliferation of fast and highly visible self-propagating worms has come to a stop. The profile of the attackers changed, and stealthier, profit-driven activities started to spread in the malicious activity scenario. For instance, in [20] it was shown how bot herders often instruct their botnets to scan within specific IP ranges, throttling down the scan frequency. Different research works [62, 50] showed how, consistently with this change in scenario, the Internet was affected by a proliferation of smaller activities highly localized.

In order to tackle this new reality different projects have proposed different data collection strategies allowing to spread the observation perspective over the IP space and characterize the locality of these phenomena.

3.3.1 Collapsar

Collapsar[96] proposes a decentralized architecture composed of a large number of honeypots deployed in different network domains. This approach tries to address the problem that centralized honeypot farm have limited view of Internet activity. The core idea of Collapsar is to deploy traffic redirectors in multiple network domains and examine the redirected traffic in a centralized farm of honeypots. This approach has the benefit that we can deploy honeypots in many networks without the need of honeypot experts on each network. All the processing and detection logic will be done in the centralized honeypot farm, also referred as Collapsar center.

An overview of the Collapsar architecture is shown at Figure 3.1. The Collapsar architecture has three parts. The first part is the traffic redirector. The traffic captures all packets and afterwards filters them, according to rules specified by the network administrator. All packets that pass the filter are encapsulated and sent to the Collapsar center. The redirection can be done either through the Generic Routing Encapsulation (GRE) tunneling mechanism of a router or using an end-system-based approach. The recirector is implemented as a virtual machine running an extended version of User-Mode Linux[67] (UML), using libpcap,libnet and specialized kernel modules.

The second part is the front-end of the Collapsar center. It receives encapsulated packets from redirectors, decapsulates them and dispatches them to honeypots of the Collapsar center. It also takes responses from honeypots and forwards them to the originating redirectors. Upon receipt, redirectors will inject the responses into their network. In that way, an attacker has the sense that communicates with a host in the network of the recirector but in reality she communicates with the Collapsar center. However, the front-end does more than packet dispatching. Its role is extended to assure that traffic from honeypots will not attacks other hosts on the Internet. To prevent such malicious activities, introduces three assurance models: logging, tarpiting and correlation. Logging module is embedded in the honeypots guest OS as well as log storage in the physical machine's host OS in order to be invisible to the attacker. The tarpiting module throttles outgoing traffic from honeypots by limiting send rate and also scrutinizes outgoing traffic based on known attack

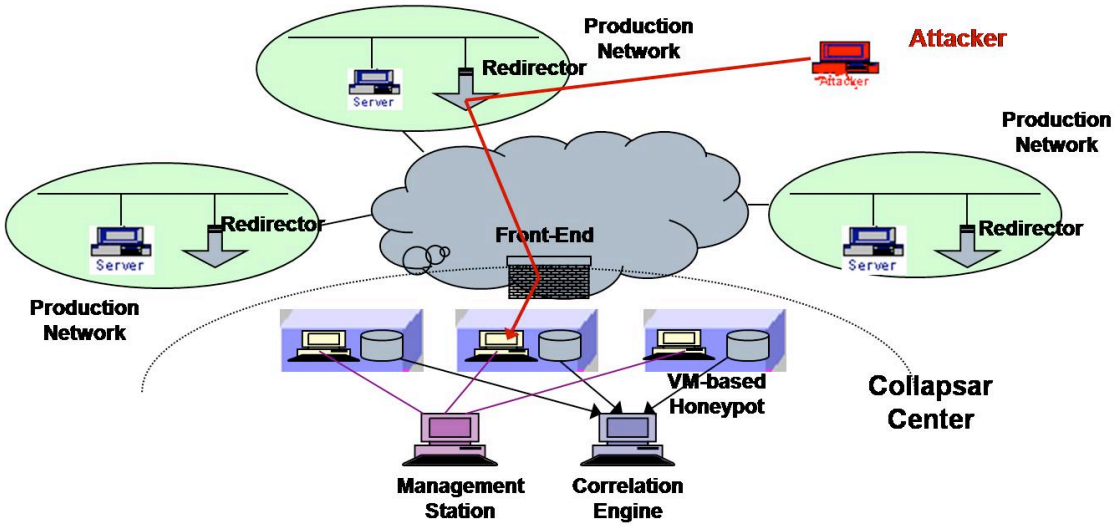

Figure 3.1: Architecture of Collapsar

signatures. Snort-inline performs this task. Snort-inline is a modified version of Snort[164], a very popular intrusion detection system. While Snort is a system that passively monitors traffic, Snort-inline intercepts on the traffic and prevents malicious traffic from being delivered to the protected network. The correlation module is able to detect network scanning by correlating traffic from honeypots that logically belong to different production networks.

The last part of the architecture is the Collapsar center. The center is a farm of high-interaction honeypots. Honeypots run services inside a virtual machine and have the same network configuration as other hosts in the production network, that is the hosts running the redirectors. Virtual machines used are VMware and UML, with UML being more preferable due to the fact that it is open-source and allows better customization, especially to network virtualization issues.

We can identify two major drawbacks in the approach proposed by Collapsar. The first one is that redirectors need a dedicated machine that communicates with a predefined set of front-ends, imposing administrative overhead for the maintenance of the redirector. Furthermore, it implies a level of trust between the redirectors and the Collapsar center Once the identity of front-ends is known, they are susceptible to direct attacks and then redirectors become useless. The second drawback is that traffic redirection adds almost double latency, according to paper measurements, that helps attackers to identify redirectors, e.g. by correlating response times from the redirector and other machines in its production network.

3.3.2 NoAH

The NoAH project[140] follows an approach similar to the Collapsar architecture. Since a centralized honeypot farm provides a limited view of network activity, the NoAH project also introduces mechanisms for deploying decentralized honeypots. The NoAH architecture is composed of two parts. The first part is the NoAH center that consists of multiple honeypot farms. Honeypot farms run high-interaction honeypots based on the Argos software. The Argos system has two major benefits: a) it allows the detection of both known and unknown (0-day) attacks and b) once the attack is detected, the faulty process is stopped by Argos. Thus, honeypot farms cannot infect other Internet hosts as vulnerable processes are stopped immediately when they are exploited. The second part of the NoAH architecture is the traffic forwarder called Honey@home. Unlike to the

Collapsar approach, where traffic redirectors are dedicated machines or GRE-enabled routers, the NoAH architecture follows a more deployable approach. The honey@home software is an end-system approach. It is a lightweight tool that runs in both Windows and Unix operating systems but does not need a dedicated machine or any specialized configuration. It silently runs in the background and claims IP addresses either through DHCP service or statically (arbitrarily large number of IP addresses), if the user configures it so. All traffic directed to the claimed IP addresses is encapsulated and sent to NoAH center. Responses from NoAH center are sent back to honey@home client, which injects them back to the attacker. Similar to Collapsar, the attacker thinks she communicates with the honey@home client but she is logically connected to a honeypot of the NoAH center. The honey@home client can also penetrate NAT environments, if the NAT router has UPnP enabled. In that case, several ports are temporarily forwarded to honey@home clients as long as they are running.

The major difference between honey@home and Collapsar redirectors is that the first protects the identity of both clients and honeypots. To achieve this, honey@home clients communicate with honeypot farms through Tor[68]. Tor is an anonymization network that is consisted of several hundreds of routers that perform onion routing. The Tor network supports both client and server anonymity by offering the so-called hidden services. The client only knows a pseudonym of the server, for example xyz.tor. When the client tries to connect to that pseudonym, the Tor network sends to both client and server a rendez-vous point. This point is a Tor router. When both client and server have connected to the rendezvous point, their connections are linked and form a full path but both of them only know that they are connected to the rendezvous point. By protecting both client and honeypot anonymity, the NoAH architecture is not vulnerable to direct attacks. However, as also in the NoAH approach traffic is forwarded, clients can be identified by latency measurements. However, the identity of honeypots will remain secret, even if most of Tor routers are compromised.

3.3.3 Potemkin

Honeypot farms usually require a large number of physical machines in order to run a few tens of virtual machines. Virtual machines in fact consume a large amount of physical memory and processing power and it is hard to run more than ten in the same physical machine. The Potemkin approach[204] proposes an architecture that overcomes this problem and improves honeypot scalability. The Potemkin architecture is based on two key observations. The first one is that most of a honeypot's processor cycles are wasted idling, as they usually wait for an adversary to connect to them. The second one is that, even when serving a request, most of a honeypot's memory is also idle.

On each physical machine, a virtual machine monitor (VMM) is running. When a packet arrives for a new IP address, the VMM spawns a new virtual machine. In this way, we have a virtual machine running only when needed, that is on a per-request basis. However, spawning a new VM for each request is an expensive operation. To reduce this overhead, the flash cloning technique is used in the Potemkin architecture. After the boot of the first VM instance, a snapshot of this environment is taken. This snapshot is then used to derive subsequent VM images. The process responsible for VM cloning is the cloning manager. It instructs Xen to create a VM based on the reference snapshot. After the VM is created and successfully resumed, the clone manager instructs the guest operating system to change its IP address based on the destination address of the request. During the cloning process the VMM stores packets destined for the VM. After cloning is finished, packets are flushed to the VM. Per-request VM cloning solves the problem of wasted processor time spent by a honeypot on waiting for requests. To overcome the problem of large memory

consumption, the delta virtualization technique is used. The notion behind delta virtualization is that most of the memory pages among VMs are common, for example pages of operating system, and thus can be shared. This technique follows the copy-on-write approach for pages that need to be changed by a VM.

3.3.4 The Leurré.com project

The Leurré.com project [111] is a distributed honeypot deployment result of the work of Pouget et al. [152]. The main purpose of this deployment is the generation of a standard distributed platform to collect *unbiased* information on network attacks. The project aims at deploying equal and thus *comparable* honeypot sensors in different locations of the IP space, and study the differences in the observations in order to ideally build a "map" of the Internet Weather.

The sensors are deployed on a partnership basis: any research entity willing to access all the collected data needs to become part of the project by installing a honeypot sensor. At the moment of writing, the project has deployed over 50 sensors, covering all the 5 continents. Each sensor, based on Honeyd, emulates the presence of 3 IPs with three different operating systems; two from the Windows family (98 and NT server) and Redhat 7.3. It monitors the network interaction of the attacking sources to these IPs through the collection of the full packet traces in *pcap* format. The emulation is purely passive, no application level script is associated to the open ports. The honeypots thus limit their emulation to the establishment of TCP connections but never reply to client requests.

In order to carry on meaningful analyses of the collected data, the honeypots logs are collected in a central database end enriched through correlation with other information sources:

- **IP Geolocation.** The project takes advantage of Maxmind [119] to retrieve information on the geographical location of the attackers and the organization responsible for the network range in the registrar database.
- **Passive OS fingerprinting.** Taking advantage of passive OS fingerprinting techniques [221] it is possible to inspect the collected packet traces and infer generic information of the operating system of the attacking client.
- **Reverse DNS resolution.** Reverse DNS resolution information is stored for every attacking source observed by the deployment.

In the Leurre.com terminology, a single host running the modified honeyd is called a platform. As honeyd emulates three operating systems, each platform needs 3 dark IP addresses to listen to. These IP addresses are consecutive and each emulated OS is assigned to listen to one of them. The reason for listening to consecutive IP addresses is to identify attackers that scan subnets. If all three emulated OSes are contacted by an attacker, it is a strong indication of scanning.

All this information is stored and aggregated in a set of meta-tables organized in such a way to allow its easy usage in data-mining tasks. Participants in the Leurre.com project need to deploy a platform and in return they are granted access to the centralized database. A detailed overview of the collected information can be found in [110].

During the Leurre.com deployment, authors have gathered some interesting statistics considering attack sources. Note that as the paper was written in 2004, the statistical results may be invalid for present time. Most of the attacks, about 80% to 95%, come from Windows machines. 35% of the machines that launched attacks are clearly identified as personal computers. The identification was done by checking the reverse DNS name for patterns like "adsl" or "dialup". Considering most

targeted ports, authors found that Windows RPC ports (135, 130 and 445) are the most popular ones.

3.3.5 Honeynets

A honeynet is an architecture proposal for deploying honeypots. Deployed honeypots can be both low- and high-interaction honeypots but honeynet architecture discusses mainly about high-interaction ones. According to the Honeynet Project[194] architecture, honeypots live in a private subnet and have no direct connectivity with the rest of the Internet. Their communication is controlled by a centralized component, actually the core of the architecture, called honeywall. Honeywall performs four operations: data capture, data collection, data analysis and data collection. Data capture mechanism monitors all traffic to and from the honeypots. The challenge here is that a large portion of the traffic is over encrypted channels (SSH, SSL, etc.). To overcome this problem, Sebek[191] was introduced to the honeynet architecture. Sebek is a hidden kernel module that captures all host activity and sends this activity to the honeywall. As Sebek runs in the host level, it can capture traffic after being decrypted. Detectability of Sebek is a challenge. Attacker must not be able that this module is running and he must not be able to see the traffic from the Sebek module to the honeywall. The detectability of Sebek is studied in [69]. The data control mechanism tries to mitigate risk from infected honeypots. As honeypots will eventually be compromised, they can be used for attacking other non-honeypot systems. Data control can be performed in many ways; limiting the number of outbound connections, removing attack vector from outgoing traffic or limiting the bandwidth. The removal of attack vectors is performed by Snort-inline. The strategy followed is specified by each organization that deploys a honeynet. Data analysis processes all the information gathered by the data capture mechanism to collect useful statistics and attack properties. Data collection applies to organizations that have multiple distributed honeynets. Its main task is to gather and combine the data. The Honeynet alliance has currently 16 members, distributed in 3 continents. The honeynet community is very active and has publish several "Know your enemy" white-papers available at project's website [192]. Honeynet farms were originally used to capture and analyze manual attacks. Recently, they have been used to track phishing attempts and botnets.

3.3.6 A hybrid honeypot infrastructure

Provos et al. in [23] propose an architecture that combines the scalability of low-interaction systems with the interactivity of high-interaction ones. The architecture consists of three components: low-interaction (or lightweight) honeypots, high-interaction honeypots and a command and control mechanism. The role of low-interaction honeypots is to filter out uninteresting traffic. Connections that have not been established (the 3-way handshake was not completed) or payloads that have been seen in the past are part of the uninteresting traffic. Low-interaction honeypots maintain a cache of payload checksums. If the payload of the first packet (after connection is established) has not been seen in the past, it is considered as interesting. According to the measurements of the paper, around 95% of the packets with payload have been observed in the past. All interesting traffic is handed off to high-interaction honeypots. The handoff mechanism is implemented by a specialized proxy. Once a packet is marked as interesting, the proxy establishes a connection with the back-end and replays this packet. Next packets of the "interesting" connection will be forwarded to the back-end by the proxy. The honeyd system is used as the main core of the low-interaction honeypots. The high-interaction honeypots run on VMware and form the back-end of the architecture. The back-end is set up not to be able to contact the outside world. Instead of

blocking or limiting the outgoing connections, traffic generated by these honeypots is mirrored back to other honeypots. As long as there are uninfected machines, the infection will spread among the honeypots, allowing the capturing of exploits and payload delivery. However, this architecture does not work for malware that downloads its code from an external source, like a web site. To detect whether high-interaction honeypots are infected, their network connections are monitored as well as changes in their filesystem. The virtual machines of infected honeypots are returned to a known good state, through the snapshot mechanism of VMware. The command and control mechanism aggregates traffic statistics from low-interaction honeypots and monitors the load of high-interaction ones. It also analyzes all data from virtual machines to detect abnormal behavior such as worm propagation. The proposed hybrid infrastructure looks similar to the infrastructures proposed by Collapsar and NoAH. However, this approach focuses more on filtering the interactions before they reach the high-interaction honeypots and additionally the backend architecture is fundamentally different as in this approach mirroring is performed.

3.3.7 Shadow honeypots

Architectures presented so far use honeypots as non-production systems, living at different network domains than production systems and listening to unused IP address space. Shadow honeypots[17] propose a different approach for detecting attacks that couples honeypots with production systems. The architecture consists of three components: a filtering component, a set of anomaly detectors and shadow honeypots. The filtering component blocks known attacks from reaching the network. Such component can be either a signature-based detector, like Snort, or a blacklist of known attack sources. The array of anomaly detectors, each one running with different settings in respect to their sensitivity and configuration parameters, is used to classify which traffic is suspicious. The traffic that is characterized as anomalous is forwarded to shadow honeypots. Their main role is to offload the shadow honeypots as much as possible by forwarding them only the traffic that may include an attack.

Shadow honeypots are cloned instances of production servers but they are heavily instrumented to detect attacks. Two types of shadow honeypots can be identified: loosely-coupled and tighty-coupled. Loosely-coupled honeypots are deployed on the same network of the protected server, running a copy of the protected applications but in a different machine without sharing state. However, the effectiveness of loosely-coupled shadow honeypots is limited to static attacks that do not require to build state at the application level. Tightly-coupled shadow honeypots run on the same machine as the protected applications and share their state. Shadow honeypots, as stated before, are instrumented versions of protected applications. The instrumentation allows the accurate detection of buffer overflow attacks and is based on the pmalloc() concept, as described in [176]. Pmalloc() is a replacement for malloc(), the standard memory allocation routine, and it works as follows. Before and after an allocated memory block requested to pmalloc(), read-only memory pages are placed. If an overflow attack is going to take place, it will try to write on the read-only pages and an exception will be thrown. The exception is caught by the pmalloc() routine, indicating the presence of an attack. (note: the concept of pmalloc() was extended to include statically allocated arrays). The major drawback of this instrumentation approach is the requirement for the application's source code.

When the shadow honeypot detects an attack, it state is rolled back as it was before the attack and the malicious content is not forwarded to the normal application. It also informs the anomaly detectors about the attack so they can tune their detection models for better performance. If the shadow honeypot does not detect any attack, the request is handled to the normal application. Again, the anomaly detectors are informed that this was not an attack so they can update their

models. Shadow honeypots can be also used to protect the client from client-side exploits, such as the buffer overflow in the JPEG handling routine of Internet Explorer. As an example, an instrument copy of Mozilla Firefox can handle web requests. According to the paper, the overhead of the instrumented version is around 20%.

3.3.8 Vigilante

Vigilante is an infrastructure that aims at worm containment. The architecture of Vigilante is based on the collaboration of end hosts and makes no assumptions that collaborating hosts trust each other. The proposed approach has preferred to move from network-level to host-level in order to eliminate problems like encrypted traffic or lack of information about software vulnerabilities. End hosts act as honeypots; they run instrumented version of software that normally wouldn't run on the host. For example, a host can run an instrumented version of a database, not a common application for a normal host. The alert generation is done using two techniques. The first uses non-executable pages around stack and heap pages to detect code injection attacks. If a buffer overflow tries to write on a non-executable page an exception is raised and caught. The second technique is the dynamic dataflow analysis. The concept of dataflow analysis is very similar to memory tainting. Data coming from the network are marked as dirty and if dirty data are going to be executed or used as critical arguments for a function, a signal for exploit is raised. Unlike approaches described in Section 2.1.3, which use specialized versions of virtual machines, Vigilante uses binary rewriting at load time. Every control transfer instruction and every data movement instruction is instrumented. A bitmap is kept to track dirty data throughout memory, one bit for each memory page. Additionally, information for where the dirty data came from are stored to help the analysis and alert generation.

The contribution of Vigilante is the concept of self-certifying alerts (SCAs). SCAs are distributed among the collaborating of hosts and their novelty is that they can be verified by recipients. This property eliminates the need for trust between the hosts. Three types of SCAs can be identified: arbitrary execution control, arbitrary code execution and arbitrary function argument, with each type covering a different type of vulnerabilities. All types of SCAs contain some common information. This information is the identity of vulnerable service, verification information to assist alert verification and a sequence of messages that contain necessary data to trigger the vulnerability. Upon received a SCA, the host uses the verification information the sequence of messages to reproduce the vulnerability. The alert distribution is done using the Pastry system. There was no real deployment and authors did simulations to evaluate the architecture. According to their results, the generation time of a SCA varies from 18 up to 2667 milliseconds, depending on the worm instance, while verification time needs up to 75 milliseconds. With a very small fraction of detectors against the total vulnerable population, infection rate can reach up to 50%. When this fraction reaches 0.001, infection rate falls to 5% and drops to nearly zero when this fraction reaches 0.01. The total population simulated was 500,000 hosts but only a subset S was vulnerable. The choice of S was done based on real data gathered during the worm outbreaks.

3.3.9 iSink

The iSink architecture[219] aims at monitoring large unused IP address space, such as /8 networks. Although this work focuses on measuring packet traffic, we will study its design and properties, which are related to honeypot infrastructures. The design model of iSink is to respond to traffic that goes to unused IP address space. However, as iSink deals with large address space, a scalable architecture is needed. Authors considered four systems that can be used as responders: Honeyd,

honeynet, LaBrea and ActiveSink. ActiveSink is a framework written by authors using the Click router language[99]. This framework includes several responders, such as ICMP, ARP, Web, SMTP, IRC and NetBIOS responders. Additionally, responders for MyDoom and Beagle backdoors were also implemented. ActiveSink responders are stateless, and still accurate, in order to achieve a high degree of scalability. Even for complex protocols, it is possible to construct a response by looking at the last request packet. Furthermore, there is need for interacting with the attacker up to the point where an attack is detected. For example, if an attack is taking place on the fifth step of a very long conversation, there is no need to emulate further than this step. The four systems were tested along five main criteria: configurability, modularity, flexibility, interactivity and scalability. Honeynet was discarded because of low configurability and medium scalability. LaBrea, on the other hand, has high scalability but very low configurability, modularity and flexibility. Honeyd presents high configurability and flexibility but medium scalability. ActiveSink, finally, is highly scalable, configurable, modular and flexible, with only drawback depending its interactivity on responders (medium interactivity according to authors). The performance of iSink architecture was evaluated using TCP and UDP packet streams at rates up to 20,000 packets per second. Each packet was a connection attempt. iSink didn't suffer from losses at all rates for both protocols. The system was also deployed in four class B networks and one class A network. The amount of traffic received in these networks was large. iSink node for class A network was receiving between 4,000 and 20,000 packets per second.

3.3.10 Internet Motion Sensor

The Internet Motion Sensor (IMS)[21] is a distributed Internet Telescope. In order to obtain visibility on localized phenomena, the Internet Motion Sensor spreads its observation points on different network blocks. It uses 28 unused IP blocks, ranging in size from /25 to /8. The Internet Motion Sensor can thus be seen as a hybrid solution, that tries to combine together the observation capabilities of Internet Telescopes with respect to global activities and achieve visibility on localized phenomena through the dispersion of the telescope address blocks along different IP blocks. The IMS architecture consists of a set of blackhole sensors. Each sensor monitors a block of unused IP addresses and has both an active and passive component. The passive component is a traffic logger that records all traffic destined to the monitored address space. The active component is a lightweight responder, aiming at raising the level of interaction with the attacker. While UDP and ICMP are stateless and no response is needed, TCP needs a connection to be established until to see the actual data of the attack. The lightweight responders establishes the incoming TCP connections and captures the payload data. The responses are application-agnostic as they do not provide any protocol emulation. While this may not work for all cases, it is enough for many cases like the Blaster worm that did not wait for any application responses. To avoid recording of unnecessary traffic, payload caching is used. The checksum of payload data is computed for each packet and if it is has been seen in the past the payload data are not stored. If not, the payload is stored along with its checksum. The observed hit rate in IMS sensors is approximately 96%, which yields to storage savings by a factor of two. Checksums also provide a convenient way to gather quick statistics about traffic and a way to filter out traffic from other components, like intrusion detection systems. The IMS systems tries to answer questions regarding worm demographics, virulence, propagation and the timescale of responses to attacks.

The lack of interaction with clients may not suffice though to identify subtle differences among different network activities: for instance, some exploits consist of several interactions between the client and the server, the first of which may be an otherwise benign query that does not reveal yet the real intention of the attacker. The techniques introduced in [21] do not appear to be adequate

to correctly handle these more complex activities, in which the attacker exposes its real intentions only after a set of preliminary interactions with the victim.

3.3.11 Honeystat

HoneyStat [64] is an effort to compliment global monitoring strategies and provide an early worm detection system by inspecting local networks. Honeystat proposes minimal honeypots created in an emulator and multihomed to cover a large address space. The emulator used is the VMware GSX server V3, which supports up to 64 isolated virtual machines on a single hardware system. Most modern operating systems support multihoming, which is assigning multiple IP addresses to a single network interface. Windows NT allow up to 32 IP addresses while most flavors of Linux up to 65536. Each virtual machine, also called node, was configured to have 32MB RAM and 770MB virtual drive. Nodes were instrumented to capture three types of events: memory, network and disk events. Memory events are considered any kind of alert by buffer overflow protection software, like StackGuard[53] or Windows logs. Each outgoing TCP SYN or UDP traffic is a network event. Disk events are generated by activities that tries to write to protected file areas. Kqueue is a monitoring tool that performs this task, although protected file areas have to be listed manually (e.g. the c:/windows/system directory or the registry file). For each the OS and patch level were also recorded as well as associated data, like stack state for memory events, packets for network events and delta of the file changes for disk events.

All recorded events are forwarded to an analysis node. If the event is a network event then the reporting honeypot is reset. This action is taken to prevent the node from attacking other machines. Besides, by the time a network activity is initiated, enough information has been recorded to study the worm infection cycle. The analysis node is also responsible to decide if some nodes should be redeployed. For example, if a worm hits a vulnerable OS, it would make sense to redeploy some of the nodes with the vulnerable OS to capture more events for the worm. Finally, each event is correlated with all other events for detection of attack patterns. Event correlation is done using logistic regression, a method which is effective for data collected in short observation windows. Network traffic from 100 /24 darknets was used to evaluate the HoneyStat approach in terms of detection accuracy. The logit analysis eliminates noise from generated events when this traffic is injected to honeystat nodes and identifies correctly all worm activities. HoneyStat evaluation, in terms of false positives, was done using attack data from the Georgia Tech Honeynet project mixed with non-attack background traffic. Attack data was from July 2002 to March 2004. There are two reasons for a honeystat node to generate a false positive: a) normal background traffic is characterized as worm and b) repeated human break-ins are identified as a worm. Honeypot events produced by nodes did not produce any false positive. However, according to authors, the false positive rate may be different than zero in a larger dataset.

3.3.12 HoneyTank

The HoneyTank project [201] aims at collecting and analyzing large amounts of malicious traffic. The data destined for the unused IP address space of a network are forwarded by cooperating network devices, like routers, and captured by an IDS called ASAX (Advanced Sequential Analyzer on Unix). The IDS emulates services on these addresses to capture data sent to services (e.g. web-server). The architecture consists of a single sensor on which ASAX is deployed and the cooperating devices. As the sensors are distributed,the authors propose to use mobile IPv4 (MIPv4) to redirect the traffic to the IDS sensor. The other alternative is to use ARP and GRE tunneling but GRE is

not fully supported by routers. To integrate a new unused IP address, the IDS sends a request for this address to the router for traffic redirection.

All packets redirected to the central sensor are captured by the libpcap library. This data is imported and analyzed by the ASAX IDS. ASAX allows the flexible declaration of rules that are applied to the captured data. Each rule declaration is written in the "*RUSSEL*" language and is used to analyze the captured packet and to apply actions that depend on the result of the prior analysis. The rule declaration allows the authors to emulate the basic behavior of the TCP protocol itself and the protocols HTTP and SMTP protocol. The advantage of the proposed approach is that no protocol state is required, increasing the scalability of the architecture (similar to iSink responders that are also stateless). The stateless emulation of protocols is done by matching regular expressions. For example, if a request matches the "GET .* HTTP/1.1" expression, a HTTP reply with code 200 is returned. However, although the level of emulation is adequate for automated programs, a manual intrusion is able to detect that services are emulated.

Authors evaluated their approach by deploying a HoneyTank prototype in their class B campus network. The ASAX IDS was configured to emulate HTTP and SMTP and to accept connections for all other TCP ports. The port and flow length distribution were calculated. HoneyTank was deployed for around 6 hours and all the traffic received from and sent by ASAX was recorded to a tcpdump trace. The trace was then given as input to a Snort system and the occurences of the attacks were measured (20 attacks were detected). Authors also compared HoneyTank with Darknet[60]. The SYN packets of HoneyTank trace were given as input to the Darknet system and the trace produced by Darknet was tested with Snort. Darknet trace contained 5 out of 20 attacks, providing less visibility to attackers than HoneyTank.

3.3.13 ARAKIS

ARAKIS [149] is a nationwide near real-time network security event early warning system developed by NASK and operated by CERT Polska. The system consists of a central repository and distributed sensors that collect and correlate data from a wide variety of sources including low-interaction honeypots, firewalls, antivirus systems and darknets. The system is oriented towards detection and characterization of new attacks based on the automated analysis of captured honeypot payloads and supporting data from other sources.

All packets coming to honeypots installed on sensors are clustered locally in real-time on each sensor based on their payload similarity. Once a sensor detects that a specific cluster has exceeded a certain size threshold, a payload-based signature of the cluster is computed and sent to the central server. The central server receives signatures from multiple sensors and periodically performs a second clusterization process. Old signatures are allocated to existing clusters, but signatures seen for the first time may form a new cluster. This new cluster, made up of signatures of new packet payloads, may therefore be representative of a new attack. All new clusters are subsequently labeled manually by operators. A public dashboard presents alarms, signatures and various statistics about malicious activity.

ARAKIS is one of the initial data sources for WOMBAT.

3.3.14 Hflow

Hflow is a data coalescing tool to support honeynet network analysis. It aggregates data from Snort, p0f, and Sebekd into a unified data structure suitable for storage in a relational database. The recently released version of Hflow offers further improvements over traditional netflow based

approaches for high interaction honeynet research and attempts to address some of the potential performance issues with Sebek and Hflow.

3.4 Malware collection initiatives

We have seen in Section 2.2 how certain honeypot deployments take advantage of different techniques to collect malware samples. The research community has an increasing interest in accessing these malware samples, in order to study the characteristics of these samples, the obfuscation techniques being used, and develop better ways to detect infections. A number of deployments chooses to focus on the sole collection of these samples and exploit them to know more about the attackers.

3.4.1 Malware Collect (Mwcollect) Alliance

The mwcollect Alliance [130] takes advantage of Nepenthes [134] honeypots hosted by different contributing partners to collect malware samples. Nepenthes is already described in depth in Section 2.1.2. The Nepenthes sensors take advantage of an ad-hoc protocol called *G.O.T.E.K.* to automatically submit any downloaded binary to the alliance database. The alliance provides a very rich dataset of malware samples, that are freely downloadable by any partner hosting a GOTEK-enabled Nepenthes sensor, but does not try to collect any rigorous information on the sources or the victims of the attacks.

3.4.2 Offensive Computing

This is another malware collection initiative [49] focused on the accumulation and sharing of malware samples for being used by users to study them and have better understanding about how to Project their IT resources. They also make use of the direct method of receiving submissions from users of the site, and malware is also available for download, resides other services like md5 hash queries and some specific malware analisys reports.

3.4.3 UploadMalware

This initiative [199], resides collecting malware, also distributes it to a list of AV vendors and security professionals. They offer a web interface for users sending the files, and they can contact the people behind the project to know about the nature of such files.

3.4.4 CyberTA

A richer and more complete dataset is offered by SRI International through the Cyber-TA project [1]. The dataset is built upon the logging capabilities of BotHunter [86] and takes advantage of a set of high interaction honeypots dynamically mapped to the addresses of a /17 network taking advantage of NAT techniques. BotHunter allows the detection of the malware infections and the successive sanitization of the hosts, and provides information on the infection stages in a simple behavioral signature of the malware sample. Once a high interaction honeypot is detected as infected, a forensic analysis takes place before its sanitization. This analysis consists in the extraction of all the filesystem modifications performed by the malware for offline analysis. The overall information provided by the deployment is very interesting, but the lack of rigorous containment limits the spread of the observation points to a single closely monitored network.

[1] `http://www.cyber-ta.org/releases/malware-analysis/public/`

Although not publicly available, some security companies uses public resurces (like lists of potentially malicious URLs published by one or other security companies, spam traps, crawlers, etc) for gathering malware and uses propietary honeyclients for gathering malware, some of them with a really high success ratio.

4 Analysis techniques for malicious code

The growing number of malware samples collected makes it a daunting task even to distinguish among different malcodes. To succeed, defenders need efficient warning and classification systems that can analyze and prioritize for them the malware they should first look at, depending on their likely impact. They must also have automated techniques to support them characterizing the threats and producing countermeasures in an automated way, as much as possible.

Enriching the collected code with such metadata might reveal insights into the origin of the code and the intentions of those that created, released or used it. We recall that the term "code" does not only comprise binary executables malware samples, but also takes into account any other element sent as part of an attack process and that share a common characteristic. They somehow aim at having certain action being performed for the benefit of an adversary, without consciously having been desired by the one executing them. The form in which the malicious code is delivered is not particularly relevant, and our techniques can be equally applied to malicious code in different forms.

Extracted metadata includes two types of information. First we need to identify specific code patterns that are characteristic for a certain class of malware, or for a particular group of attacks. For this purpose the focus is on data regarding the actions, or behavior, of the code as well as its structure, to understand whether two code samples are related according to their (shared) structure or similar behavior. Secondly, we are interested in capturing the "context" in which the code sample was collected, in order to be able to analyze where the collected code came from, and under which circumstances it was delivered.

Among the research efforts we reviewed there are approaches that have been used previously to solve security-related problems.

4.1 Simulation-based verification of malicious behaviors

Simulation-based verification gathers basically any black-box procedure deployed for dynamic analysis [95]. The current execution path is collected as a sequence of discrete events through the different collection infrastructures presented in the previous section (Section 3). These events are analyzed to finally be compared to the reference behavioral model: the behavioral signature. In order to characterize malware, we recall that these approaches must be enabled to extract a characteristic subset that captures the unique (and malicious) behavior of a certain program. To our knowledge, various techniques of analysis and comparison can be considered as we will describe in the coming sub-sections.

4.1.1 Expert system

Such systems "embed" the experience and expertise of an analyst through case-based rules modelling [66]. Rules are defined for each known suspicious activity. The target and the privilege level of the caller are important factors because they often draw the distinction between a legitimate action and a malicious one. The class of complexity for the rule-matching algorithms is acceptable, being equivalent to pattern matching algorithms.

The decision whether a behavior is malicious or not must then be preemptively taken, in order to react consequently before these attempts are resolved: these proactive systems are often referred to as "behavioral blockers" [133]. It must be underlined that expert systems are prone to false positives.

4.1.2 Heuristic engines

System calls and associated parameters are usually collected using a sandbox and they are analyzed as a sequential stream of events. A basic heuristic engine can be decomposed into three parts [168, 169]:

- an *association mechanisms* is used to label the different *atomic* behaviors of malware using either a weight-based or a flag-based labelling technique [202, 225]. The former uses quantitative values to express the action severity, while the latter use semantic symbols to mark a corresponding functionality.
- A *rule database* defines the detection criteria. Weight-based systems a threshold is used to decide. Otherwise, the detection rules consist in flag sequences.
- In the case of a weight-based association, *the detection strategy* is the accumulation function, chosen to correlate the captured values. In the case of flag-based techniques, several kinds of algorithm exist: greedy without possible back-steps during exploration, genetic, taboo or simulated annealing with conditioned back-steps [85].

4.1.3 State machines

Just like heuristic engines, state machines are based on sequential models of system calls. The malicious behaviors are described as Deterministic Finite Automata (DFA) according to the following principle [41, 173]:

- The states S of an automaton corresponds to the internal states of the malware along their lifecycle,
- The set of input symbols Σ defined upon the collected data which are mainly system calls,
- The transition function T describes the symbol sequences known as suspicious,
- The initial state s_0 corresponds to the beginning of the analysis,
- The set of accepting states A conveying the detection of a suspicious behavior.

From an initial state, the automaton will progress step-by-step by evaluating the elements from the sequence of collected data. If during its progression, the automaton reaches an accepting state, a malicious behavior has been discovered. Otherwise, if the automaton does not reach an accepting step before the end of the data sequence or reaches an error state, only behaviors supposed legitimate have been captured. Figure 9 gives an example of automaton detecting a file infection mechanism. In state machines, the matching algorithm is defined by the word acceptance problem by an automaton. Using deterministic finite automata, the complexity of this problem remains in P [90].

Notice that state machines can also be used for the opposite approach of behavioral detection that is to say modelling legitimate behaviors. The considered automaton is then no longer deterministic

but probabilistic. The probabilities of the different transitions may be based on the frequency of certain system call sequences during a healthy execution [120]. Unfortunately, the model put forward is used to detect macroviruses and consequently targets a specific type of application: office software. It remains almost impossible to extend generically legitimate models to every application.

4.1.4 Learning algorithms

The automatic generation of behavioral signatures is crucial to avoid the shortcomings of the simple byte signatures. For the sake of generality, we will refer to such signature generator as "rule learning algorithms": the system is first ran against a learning pool made up of large sets of malware and legitimate samples already labelled as malicious or benign. Like any learning process, the generation of behavioal signatures remains very sensitive to noise injection in the training pool. Several attacks have already been proven to be effective against similar worm signature generators [146]. During the training phase, the learning procedure tries to extract common properties between the different considered samples. Generally speaking, the extraction of such "common properties" is based on three major paradigms which are briefly described in the following.

Rules induction: This first paradigm specifies the belonging conditions for the different classes of behavior. For each sample received by the classifier, it integrates or removes certain characteristic data in the condition in order to preserve the class consistency [108, 172, 207]. Such rules are often represented as boolean expression or as decision trees [100].

Bayesian statistics: The second paradigm based on statistics is used in Bayesian-like classifiers. For each collected characteristic, the probability of finding it in a given class of malware is estimated [172, 207, 100]. Only the most stochastically significant results are kept, for instance by ensuring minimal overlapping of the characteristics in the different classes. The ideal case would obviously be when a characteristic exists with a probability of 100% in a unique class whereas it is absent of any other.

Clustering: During the learning procedure, average profiles are built for each class of malware. When deployed, the classifiers computes the value of a distance function between the profiles and the tested programs [107]. Each sample is classified into the most similar profile, in terms of the distance function. The distance function may vary from a system to another and it heavily impacts the classification accuracy.

As an example, the approach suggested in [107] analyzes the sapmles by capturing their *system calls*, and then relies on machine learning techniques to identify calls that indicate malicious intent, and flag them accordingly.

4.2 Formal verification of program structure

Since malware actions are originally written down in the code, thereby, malware can also be discovered formally through static analysis. Formal verification, in the context of detection, consists in verifying that a program abstraction satisfies or not a malicious formal specification, which is basically a bisimulation problem [95]. Thanks to this white box approach, these detectors can combinatorially explore the different execution paths. Once again different verification methods can be deployed.

4.2.1 Graph-based signatures

Isomorphism of annoted graphs uses *Control Flow Graphs* (CFG) thus works exclusively with static extraction. The assembly code instructions are abstracted by labelling the nodes of the extracted graphs. The labelling may be performed either by translating instructions to semantic labels [46, 159] or by reducing them to their basic class of operation (e.g., arithmetic, logic, function call) [32, 103]. The behavioral signature is represented by a graph structure using an annotation mechanism. Without going into details, given a generated behavioral signature graph, detection is achieved by checking whether a program satisfies a given graph template in terms of isomorphism.

Theoretically, detection algrorithms are in the class of NP-complete but its complexity can often be reduced in the particular context. In fact, CFG nodes, except in the case of indirect jumps and function returns, have a bounded number of successors, typically one or two. Isomorphism remains very sensitive to mutation techniques and in particular to any modification impacting the graph resulting from the extraction: code permutation or injection (dead code hidden behind opaque predicate, additional intermediate variables). These transformations can partially be addressed by optimization techniques developed for compilers [148, 33, 32]. The ultimate goal would be to reach a canonical and minimal form for malware, to revert most mutation effects.

We also cite the models proposed by [44, 103] that can used to detect malware instances that share a common code base, or to construct distance metrics for discriminating among malwares with different structure, and thus coming from different areas.

4.2.2 Equivalence by reduction

This is an algebraic approach working by deduction using logical equivalence at each reasoning step [214, 215].

The original program is first represented using a formal specification of the processor instruction set; this representation is constructed in order to decrease the differences between equivalent functionalities. A single algebraic expression will stand for several equivalent instructions such as `mov` operations using different registers for example. The abstract specification is then simplified by reduction using rewriting rules preserving some equivalence and semi-equivalence properties. Basically, equivalent expressions have an identical effect on the whole memory whereas semi-equivalent ones only preserve specific variables and locations. The purpose is the reduction of the number of syntactic variants.

The resulting representations are compared with known malware specifications. The complexity of the comparison algorithm is equivalent to the halting problem, thus this technique can only be deployed on limited code samples from malware.

4.2.3 Model checking

In model checking approaches, behavioral signatures are defined by means of temporal logic formulae [28, 177], an extension of the first-order logic. The model checker engine is then used to find all the intermediate states of the execution paths that satisfy such formulae. The major issue is that these algorithms enumerate all the possible execution paths which can be infinite. Symbolic temporal model checkers exist which prove to be PSpace-complete [171].

To construct more abstract signatures, in the most recent logics, registers, free variables and constants are referenced as generic values [98]; this allows to cope to code mutations by reassignment. Moreover, the temporal predicates used to explore the different execution paths have been proven to be really to detect dead code insertion and reordering.

4.3 Deviation measurement from legitimate behaviors

This third approach is the opposite of the two previous ones. Instead of defining what is a malicious behavior or structure, we will define what is a legitimate program and check dynamically that executed programs comply with the model. Host-based intrusion detection algorithms often adopts behavioral models to protect the program against code-injection attacks, by comparing the program execution with the expected behavior. This approcah resuse some of the techniques previously described: static analysis and CFG building for specification and automata for real-time checking. [77] [75, 78]

-Modelling the accepted behavior of a process through specifications: [205] or with context sensitive automaton [83]

From the WOMBAT perspective, such algorithms are useful but we recall the need for stochastic extensions of such models since they are no robust against code obfuscation or other more complex mutation techniques, often used by malware authors to circumvent this type of analyses.

4.4 Exploiting of contextual information

Contextual information, which is information describing the context in which a given event, or a series of events, have been observed, is getting used by a number of intrusion detection correlation systems to discriminate between anomalous yet harmless traffic and nefarious anomalies [104]. Contextual information was also recently used in a system that has goals closer to the WOMBAT project. In the Leurré.com project [155], such information is used to help grouping together traces of attacks that, at first glance, have nothing in common. It is only the added information provided by the context (such as the country of origins of the attacks, their timing, or their targets) that enables researchers to highlight the existence of strong correlation between these attacks. Similarly to this approach, we will investigate the important contextual elements that one can take into consideration for the various kinds of sensors that we are going to use.

Additionally, the growing trends of modular malicious code, multi-stage downloads, and increasing stealth and persistence of adversaries are beginning to defeat techniques for binary analysis. With these techniques, the whole binary does not arrive at once. Rather, in such cases, it arrives incrementally. Sometimes hiding techniques are used to hide the first portion of an attacker's toolkit before the second portion is transmitted. Also, the different pieces may be transmitted via different protocols from different sources. In this context, the attacker's full footprints are often only revealed in full within the compromised system, and, even then, pieces may be scattered in different corners of the system. For these reasons, Symantec will pursue memory image analysis. By developing statistical models of phylogenetic components of normal memory images and by leveraging to develop clustering and automated classification techniques for similarly characterizing malicious components scattered within a system, we hope to more reliably detect malicious software that has penetrated defenses and only runs quietly intermittently.

5 Conclusions and future work roadmap

The shifting paradigm of Internet attacks (which seem to have moved from servers to clients [19] and from fast spreading worms to commercially more interesting activities like theft, fraud, phishing, ...), creates a huge gap of knowledge between what we believe to be happening, and what we actually, scientifically know and can prove. Even if rumors and stories circulate within the security community about the origins, causes, consequences of these new malicious activities, very few claims can be backed up by scientific evidence.

Many issues remain open, including attribution (who is behind the current attacks? Where do they originate?), organization of cybercrime (how many organizations control the botnets? What are the trends in cyber-crime?) and technical questions (what is the spreading pattern over time? Are firewalls useful, or are attacks layered over well-known ports that are unprotected by firewalls?). The answers to such questions are extremely important, as they help decision makers to invest in the appropriate security measures.

To obtain such answers, we need data. In Chapter 2 we reviewed a number of powerful tool for collecting data on malicious threats. High interaction honeypots that allow to retrieve in-depth information on malicious threats, providing the maximum possible level of interactivity to attackers, but posing containment problems that can be addressed and *partially* solved taking advantage of different containment techniques. These containment techniques require the deployment of complex infrastructures, complexity to be added to the resource cost of the usage of real OS implementations. Low interaction honeypots, on the other hand, require a significantly lower level of complexity, but fall into another pitfall. For a low interaction honeypot to be a useful measurement tool, it needs to emulate the behavior of a real host and its protocols. Only through application protocol emulation it is possible to lure the attacker in revealing his intent and thus its nature. We have described promising research directions to address these shortcomings (such as ScriptGen, Section 2.1.6).

A number of collection infrastructures, reviewed in Chapter 3, use these building blocks to collect data. Unfortunately, very few efforts are made to share the collected information, except for some anecdotal challenges between security experts like in [193]. As can be seen by the recent requests received by ENISA [39, 73] from the European Commission, it is also widely recognized that although large amounts of security data are being gathered by several different sources, these data are frequently incomparable, if not incompatible, making their meaningful use by decision makers difficult, it not impossible. If it is so difficult for large companies to share data on incidents and consumer confidence, it is incredibly more difficult to share datasets that could entail competitive advantages.

However, besides the complexity in coordinating data collection efforts, raw data in itself is almost useless. First, such raw data is overwhelming. For instance, companies like VirusTotal and Symantec receive hundreds of thousands of seemingly unique malware samples per week. The systematic analysis of all these malware would be outrageously expensive. As a result, prioritization, grouping and automated pre-analysis of such data is needed. Many of the technical issues related to these tasks are still difficult to solve, as we discussed in Chapter 4. In fact, we do not lack data *per se*. Rather, we lack *meaningful data*. In addition to collecting malware and exploits from as many diverse sensors as possible, we need new techniques for classifying, clustering and correlating the gathered information in order to guide and help the analysis process of the most important threats

first. This a very complex problem, that can be described in general terms as the annotation of attack information with *metadata*, one of the core tasks of the WOMBAT project.

A further problem, which will be addressed by WOMBAT, is the creation of new global analysis techniques. We define as global analysis techniques the ones that can produce a precise analysis of the modus operandi of the attackers, or may reveal patterns that allow to group apparently different attacks. Conceptually speaking, this problem comes down to mining a very specific dataset, made of attack traces. Current analysis techniques do not allow to discover and to automatically extract new relevant knowledge about the attack processes on the Internet that can be validated following a rigorous and scientific methodology. Recently, Eurécom has investigated the usability of a graph based approach to group together traces that share some specific features. This technique, described in [152] relies on the iterative extraction of dominant sets of maximally similar nodes in a graph [30, 145] and has produced some preliminary results [156]. Another objective of such global analysis techniques would be to produce *predictive* models of the malicious activities. This, in the end, would help in building a new generation of more efficient early warning systems.

Bibliography

[1] Bluetooth standard. Available online at `http://standards.ieee.org/getieee802/download/802.15.1-2005.pdf`.

[2] GNU GCC online documentation on trampolines. Available online at `http://gcc.gnu.org/onlinedocs/gccint/Trampolines.html`.

[3] Home page of "LaBrea". Available online at `http://labrea.sf.net/labrea-info.html`.

[4] Home page of "The HoneyWall Project". Available online at `https://projects.honeynet.org/honeywall/`.

[5] Honeyd scripts collection. Available online at `http://www.honeyd.org/contrib.php`.

[6] Honeynet.br. Available online at `http://www.honeynet.org.br`.

[7] Ieee 802.11 standard. Available online at `http://standards.ieee.org/getieee802/download/802.11-2007.pdf`.

[8] An investigation into the unauthorised use of 802.11 wireless local area networks. Available online at `http://http://esm.cis.unisa.edu.au/new_esml/resources/publications/an%20investigation%20into%20the%20unauthorised%20use%20of%20802.11%20wireless%20local%20area%20networks.pdf`.

[9] The new zealand honeynet project. Available online at `http://newzealand.honeynet.org`.

[10] The philippine honeynet project. Available online at `http://www.philippinehoneynet.org`.

[11] Ren-isac monitoring data. Available online at `http://www.ren-isac.net/monitoring.html`.

[12] Samba. Available online at `http://www.samba.org`.

[13] Wikipedia, "honeypot (computing)". Available online at `http://en.wikipedia.org/wiki/Honeypot_(computing)`.

[14] A. Ho, M. Fetterman, C. Clark, A. Warfield, and S. Hand. Practical taint-based protection using demand emulation. In *EuroSys, Leuven, Belgium*, April 2006.

[15] Active Threat Level Analysis System (ATLAS). Home page of "Active Threat Level Analysis System (ATLAS)". Available online at `http://atlas.arbor.net`.

[16] P. Akritidis, E. P. Markatos, M. Polychronakis, and K. D. Anagnostakis. Stride: Polymorphic sled detection through instruction sequence analysis. In *Proceedings of the 20th IFIP International Information Security Conference (IFIP/SEC 2005)*, 2005.

[17] K. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14th Usenix Security Symposium*, Aug. 2005.

[18] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, E. M. K. Xinidis, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proc. of the 14th USENIX Security Symposium*, August 2005.

[19] M. Arnone. Sans: Cybercriminals targeted popular applications, network systems in 2005. *Federal Computer Week*, November 2005.

[20] P. Bächer, T. Holz, M. Kötter, and G. Wicherski. Know your enemy: Tracking botnets. Available online at `www.honeynet.org/papers/bots`.

[21] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The internet motion sensor: A distributed blackhole monitoring system. In *12th Annual Network and Distributed System Security Symposium (NDSS)*, February 2005.

[22] M. Bailey, E. Cooke, F. Jahanian, N. Provos, K. Rosaen, and D. Watson. Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic. In *Proceedings of the Internet Measurement Conference (IMC) 2005*, 2005.

[23] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos. A hybrid honeypot architecture for scalable network monitoring. Technical report cse-tr-499-04, Department of Electrical Engineering, University of Michigan, October 2004.

[24] G. Bakos. Tiny honeypot - resource consumption for the good guys. Available online at `http://www.alpinista.org/thp/`.

[25] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Visualization. In *Proceedings of the Symposium on Operating Systems Principles (SOSP '03)*, Oct. 2003.

[26] M. Beddoe. Home page of the "Protocol Informatics Project". Available online at `http://www.4tphi.net/~awalters/PI/PI.html`.

[27] F. Bellard. Qemu, open source processor emulator. Available online at `http://bellard.org/qemu/`.

[28] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi. Static detection of malicious code in executable programs. In *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS)*, 2001.

[29] K. Biba. Integrity considerations for secure computer systems. In *MITRE Technical Report TR-3153*, 1977.

[30] I. Bomze, M. Pelillo, and V. Stix. Approximating the maximum weight clique using replicator dynamics. *Neural Networks, IEEE Transactions on*, 11(6):1228–1241, November 2000.

[31] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards automatic generation of vulnerability-based signatures. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 2–16, Washington, DC, USA, 2006. IEEE Computer Society.

[32] D. Bruschi, L. Martignoni, and M. Monga. Detecting self-mutating malware using control-flow graph matching. In *Proceedings of the Conference on the Detection of Intrusions and Malwares and Vulnerability Assessment (DIMVA)*, pages 129–143, 2006.

[33] D. Bruschi, L. Martignoni, and M. Monga. Using code normalization for fighting self-mutating malware. In *Proceedings of the International Symposium on Secure Software Engineering*, pages 37–44. IEEE CS Press, 2006.

[34] bulba and Kil3r. Bypassing Stackguard and Stackshield. *Phrack Magazine*, 10(56), January 2000.

[35] C. Cowan and S. Beattie and J. Johansen and P. Wagle. PointGuard: Protecting pointers from buffer overflow vulnerabilities. In *Proc. of the 12th USENIX Security Symposium*, pages 91–104, August 2003.

[36] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle and Q. Zhang. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proc. of the 7th USENIX Security Symposium*, 1998.

[37] C. Cowan, M. Barringer, S. Beattie and G. Kroah-Hartman. FormatGuard: Automatic protection from printf format string vulnerabilities. In *In Proc. of the 10th Usenix Security Symposium*, August 2001.

[38] CAIDA Project. Home page of the caida project. Available online at `http://www.caida.org`.

[39] C. Casper. Data on security incidents and consumer confidence – you cannot get hold of it for love or money! *ENISA Quarterly*, 3(1):13–14, Jan–Mar 2007. Available online at `http://www.enisa.europa.eu/doc/pdf/publications/enisa_quarterly_03_07.pdf`.

[40] C. C. R. Center. Cybercrime is an organized and sophisticated business. available onlin at `http://www.crime-research.org/news/20.02.2006/1835/`, Feb 2006.

[41] B. L. Charlier, A. Mounji, and M. Swimmer. Dynamic detection and classification of computer viruses using general behaviour patterns. In *Proceedings of the Virus Bulletin Conference*, 1995.

[42] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[43] R. Chinchani and E. Berg. fast static analysis approach to detect exploit code inside network flows. In *In Recent Advances in Intrusion Detection*, Seattle, WA, September 2005.

[44] M. Christodorescu and S. Jha. Static Analysis of Executables to Detect Malicious Patterns. In *Usenix Security Symposium*, 2003.

[45] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. In *Security and Privacy Conference*, Oakland, CA, May 2005.

[46] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantic-aware malware detection. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 32–46, 2005.

[47] F. Cohen. A note on the role of deception in information protection. *Computers & Security*, 17(6):483–506, 1998.

[48] Computer Network Defence Ltd. Home page of "Talisker Computer Network Defense Operational Picture". Available online at `http://www.securitywizardry.com/radar.htm`.

[49] O. Computing. Home page of "Offensive Computing". Available online at `http://www.offensivecomputing.net`.

[50] E. Cooke, M. Bailey, Z. M. Mao, D. Watson, F. Jahanian, and D. McPherson. Toward understanding distributed blackhole placement. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 54–64, New York, NY, USA, 2004. ACM Press.

[51] J. A. Coret. Home page of "Kojoney". Available online at `http://kojoney.sourceforge.net`.

[52] J. Corey. Advanced Honey Pot Identification and Exploitation. Available online at `http://www.ouah.org/p63-0x09.txt`.

[53] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *Proceedings of the 7^{th} USENIX Security Conference*, pages 63–78, jan 1998.

[54] J. Crandall, F. Chong, and S. Wu. Minos: Architectural Support for Protecting Control Data. In *Transactions on Architecture and Code Optimization (TACO). Volume 3, Issue 4*, Dec. 2006.

[55] J. R. Crandall and F. T. Chong. Minos: Control data attack prevention orthogonal to memory model. In *Proc. of the 37th annual International Symposium on Microarchitecture*, pages 221–232, 2004.

[56] W. Cui. *Automating Malware Detection by Inferring Intent.* PhD thesis, University of California, Berkeley, Fall 2006.

[57] W. Cui, V. Paxson, and N. Weaver. Gq: Realizing a system to catch worms in a quarter million places. Technical report, ICSI Tech Report TR-06-004, September 2006.

[58] W. Cui, V. Paxson, N. Weaver, and R. Katz. Protocol-independent adaptive replay of application dialog. In *Proceedings of the 13th Annual Symposium on Network and Distributed System Security (NDSS)*. The Internet Society, 2006.

[59] T. Cymru. Team cymru. Available online at `http://www.team-cymru.org`.

[60] T. T. Cymru. Home page of "The team cymru darknet" project. Available online at `http://www.cymru.com/Darknet`.

[61] D. Dagonand, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine and Henry Owen. HoneyStat: Local worm detection using honeypots. In *In Proc. of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[62] M. Dacier, F. Pouget, and H. Debar. Honeypots, a practical mean to validate malicious fault assumptions. In *Proceedings of the 10th Pacific Ream Dependable Computing Conference (PRDC04)*, Tahiti, February 2004.

[63] M. Dacier, F. Pouget, and H. Debar. Towards a better understanding of internet threats to enhance survivability. In *Proceedings of the International Infrastructure Survivability Workshop 2004 (IISW'04)*, Lisbonne, Portugal, December 2004.

[64] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honeypots. In *Proceedings of the* 7^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 39–58, Oct. 2004.

[65] W. de Bruijn, A. Slowinska, K. van Reeuwijk, T. Hruby, L. Xu, and H. Bos. Safecard: a gigabit ips on the network card. In *Proceedings of 9th International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, pages 311–330, Hamburg, Germany, September 2006.

[66] M. Debbabi. Dynamic monitoring of malicious activity in software systems. In *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS)*, 2001.

[67] J. Dike. *User Mode Linux*. Prentice Hall, 2006.

[68] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the* 13^{th} *Usenix Security Symposium*, Aug. 2004.

[69] M. Dornseif, T. Holz, and C. Klein. NoSEBrEaK - Attacking Honeynets. In *Proceedings of the 2004 Workshop on Information Assurance and Security*, June 2004.

[70] DShield. Dshield distributed intrusion detection system. Available onilne at `http://www.dshield.org`.

[71] E. G. Barrantes, D.H. Ackley, S. Forrest, T. S. Palmer, D. Stefanovix and D.D. Zovi. Randomized instruction set emulation to disrupt code injection attacks. In *In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 281–289, October 2003.

[72] eEye. eeye industry newsletter. `http://www.eeye.com/html/resources/newsletters/versa/VE20070516.html`, May 2007.

[73] ENISA. Data collection of security incidents and consumer confidence. `http://www.enisa.europa.eu/pages/data%5Fcollection/`.

[74] H. Feng, J. Giffin, Y. Huang, S. Jha, W. Lee, and B. Miller. Formalizing sensitivity in static analysis for intrusion detection. In *Proceedings the IEEE Symposium on Security and Privacy*, Oakland, CA, 2004.

[75] H. Feng, J. Giffin, Y. Huang, S. Jha, W. Lee, and B. Miller. Formalizing sensitivity in static analysis for intrusion detection. In *IEEE Symposium on Security and Privacy*, 2004.

[76] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *Proceedings of the IEEE Security and Privacy Conference*, Oakland, CA, 2003.

[77] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *IEEE Symposium on Security and Privacy*, 2003.

[78] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 120, Washington, DC, USA, 1996. IEEE Computer Society.

[79] G. E. Suh, J. W. Lee, D. Zhang and S. Devadas. Secure program execution via dynamic information flow tracking. *ACM SIGOPS Operating Systems Review*, 38(5):86–96, December 2004. SESSION: Security.

[80] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *In Proc. of the ACM Computer and Communications Security (CCS) Conference*, pages 272–280, October 2003.

[81] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai and P. M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *In Proc. of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[82] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *In Proc. of the 10th ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, February 2003.

[83] J. Giffin, S. Jha, and B. Miller. Efficient context-sensitive intrusion detection. In *Network and Distributed System Security Symposium (NDSS)*, 2004.

[84] J. T. Giffin, S. Jha, and B. P. Miller. Automated discovery of mimicry attacks. In *RAID*, pages 41–60, 2006.

[85] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*. Springer, 2003.

[86] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium*, August 2007.

[87] L. Halme and R. Bauer. AINT misbehaving-a taxonomy of anti-intrusion techniques. *Proceedings of the 18th National Information Systems Security Conference*, pages 163–172, 1995.

[88] A. B.-B. Hank Nussbacher. Home page of "The IUCC/IDC Internet Telescope". Available online at `http://noc.ilan.net.il/research/telescope/`.

[89] A. Ho, , M. Fetterman, C. Clark, A. Warfield, and S. Hand. Practical Taint-Based Protection using Demand Emulation. In *Proceedings of the EuroSys*, pages 39–58, Apr. 2006.

[90] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages and Computation, Second Edition*. Addison Wesley, 1995.

[91] K. Hyang-Ah and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. of the 13th USENIX Security Symposium*, 2004.

[92] iDefense. Home page of "Multipot". Available online at `http://labs.idefense.com/files/labs/releases/previews/multipot`.

[93] J. C. Rabek, R. I. Khazan, S. M. Lewandowski and R. K. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. In *In Proc. of the ACM workshop on Rapid Malcode*, 2003.

[94] J. Etoh. GCC extension for protecting applications from stack-smashing attacks. Technical report, IBM, June 2000.

[95] G. Jacob, H. Debar, and E. Filiol. Behavioral detection of malware: From a survey towards an established taxonomy. *Journal in Computer Virology*, 4(Coming WTCV'07 Special Issue), 2008.

[96] X. Jiang and D. Xu. Collapsar: A VM-Based Architecture for Network Attack Detention Center. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, Aug. 2004.

[97] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):443–471, 2003.

[98] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith. Detecting malicious code by model checking. *Lecture Notes in Computer Science*, 3548:174–187, 2005.

[99] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. In *ACM Transactions on Computer Systems 18(3)*, pages 263–297, Aug. 2000.

[100] J. Kolter and M. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 470–478. ACM Press, 2004.

[101] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *2nd Workshop on Hot Topics in Networks (HotNets-II)*, 2003.

[102] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Automating mimicry attacks using static binary analysis. In *14th Usenix Security Symposium*, Baltimore, MD, August 2005.

[103] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

[104] C. Kruegel and W. Robertson. Alert Verification - Determining the Success of Intrusion Attempts. In *Workshop on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2004.

[105] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 201–208, New York, NY, USA, 2002. ACM Press.

[106] C. Krügel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In A. Valdes and D. Zamboni, editors, *RAID*, volume 3858 of *Lecture Notes in Computer Science*. Springer, 2005.

[107] T. Lee and J. Mody. Behavioral Classification. In *15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, 2006.

[108] W. Lee, S. Stolfo, and P. Chan. Learning patterns from unix process execution traces for intrusion detection. In *Proceedings of the AAAI97 workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. Addison Wesley, 1997.

[109] C. Leita, K. Mermoud, and M. Dacier. ScriptGen: An Automated Script Generation Tool for Honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, Dec. 2005.

[110] C. Leita, V. H. Pham, O. Thonnard, E. Ramirez-Silva, F. Pouget, E. Kirda, and M. Dacier. The Leurre.com Project: Collecting Internet Threats Information using a Worldwide Distributed Honeynet. In *Proceedings of the 1st Wombat Workshop*, 2008.

[111] Leurre.com. Home page of "Leurre.com Honeypot Project". Available online at `http://www.leurrecom.org`.

[112] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: a basis for building self-protecting servers. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 213–222, Alexandria, VA, USA, 2005. ACM Press.

[113] T. Liston. Home page of "LaBrea". Available online at `http://labrea.sourceforge.net`.

[114] M. E. Locasto, S. Sidiroglou, and A. D. Keromytis. Application communities: Using monoculture for dependability. In *Proceedings of the 1st Workshop on Hot Topics in System Dependability (HotDep)*, pages 288 – 292, Yokohama, Japan, June 2005.

[115] M. E. Locasto, A. Stavrou, G. F. Cretu, and A. D. Keromytis. From stem to sead: Speculative execution for automated defense. In *Proceedings of the 2007 USENIX Annual Technical Conference*, pages 219–232, 2007.

[116] G. Lyon. Home page of "Nmap". Available online at `http://insecure.org/nmap`.

[117] M. Costa and J. Crowcroft and M. Castro and A Rowstron, L. Zhou and L. Zhang and P. Barham. Vigilante: End-to-end containment of internet worms. In *In Proc. of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, Brighton, UK, October 2005.

[118] M. Frantzen and M. Shuey. StackGhost: Hardware facilitated stack protection. In *In Proc. of the 10th USENIX Security Symposium*, pages 55–66, August 2001.

[119] Maxmind. IP geolocation and online fraud prevention. Available online at `http://www.maxmind.com/`.

[120] G. Mazeroff, V. D. Cerqueira, J. Gregor, and M. G. Thomason. Probabilistic trees and automata for application behavior modeling. In *Proceedings of the 43rd ACM Southeast Conference*, 2003.

[121] McAfee Inc. Home page of "SiteAdvisor". Available online at `http://www.siteadvisor.com`.

[122] McAfee Inc. Last "SiteAdvisor" report release. Available online at `http://www.mcafee.com/us/about/press/corporate/2008/20080604_181010_g.html`.

[123] L. McVoy. LMbench - Tools for Performance Analysis. Available online at `http://www.bitmover.com/lmbench`.

[124] Microsoft Corporation. Microsoft Security Bulletin MS04-028. Available online at `http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx`.

[125] Microsoft Corporation. Microsoft Security Bulletin MS06-001. Available online at `http://www.microsoft.com/technet/security/bulletin/MS06-001.mspx`.

[126] D. Moore. Network telescopes: Observing small or distant security events. *Proceedings of the 11th USENIX Security Symposium*, 2002.

[127] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[128] M. Moser. Homepage of hotspotter. Available online at `http://www.remote-exploit.org/codes_hotspotter.html`.

[129] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A crawler-based study of spyware in the web. In *Proceedings of the 13th Annual Symposium on Network and Distributed System Security (NDSS)*. The Internet Society, 2006.

[130] mwcollect Alliance. Home page of "mwcollect - Malware Collection Made Easy". Available online at `http://www.mwcollect.org`.

[131] myNetWatchman. mynetwatchman - network intrusion detection and reporting. Available online at `http://www.mynetwatchman.com`.

[132] N. Dor, M. Rodeh, and M. Sagiv. CSSV: Towards a realistic tool for statically detecting all buffer overlows in C. In *In Proc. of the ACM Conference on Object-Oriented Programming, Systems, Languages and Application*, October 2003.

[133] C. Nachenberg. Behavior blocking: The next step in anti-virus protection, 2002.

[134] Nepenthes Development Team. Home page of "Nepenthes". Available online at `http://nepenthes.mwcollect.org`.

[135] A. Networks. Terms of use of atlas data. Available online at `http://atlas.arbor.net/about/terms`.

[136] J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: automatic protocol replay by binary analysis. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 311–321, New York, NY, USA, 2006. ACM Press.

[137] J. Newsome, B. Karp, and D. X. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, May 2005.

[138] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the* 12^{th} *Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

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

[140] NoAH. Home page of "Network of Affined Honeypots (NOAH)". Available online at `http://www.fp6-noah.org`.

[141] N. I. of Information and C. T. (NICT). Home page of the NICTER project. Available online at `https://www2.nict.go.jp/y/y211/e_index.html`.

[142] L. Oudot. Wireless honeypot countermeasures. Available online at `http://www.securityfocus.com/infocus/1761`.

[143] M. Overton. Worm charming: taking smb lure to the next level. *Virus Bulletin Conference*, 2003.

[144] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *Proceedings of the 4^{th} ACOM SIGCOMM conference on Internet measurement*, pages 27–40, 2004.

[145] M. Pavan and M. Pelillo. A new graph-theoretic approach to clustering and segmentation. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 145 – 152, June 2003.

[146] R. Perdisci, D. Dagon, P. W. L. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings of IEEE Symposium on Security and Privacy*, 2006.

[147] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, May 2006.

[148] F. Periot. Defeating polymorphism through code optimization. In *Proceedings of the Virus Bulletin Conference*, pages 142–159, 2003.

[149] N. Polska. Home page of the "ARAKIS Project". Available online at `http://www.arakis.pl`.

[150] G. Portokalidis and H. Bos. Eudaemon: Involuntary and on-demand emulation against zero-day exploits. In *Proceedings of ACM SIGOPS EUROSYS'08*, pages 287–299, Glasgow, Scotland, UK, April 2008. ACM SIGOPS.

[151] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proc.of the 1st ACM SIGOPS EUROSYS*, Leuven. Belgium, April 2006.

[152] F. Pouget. *Distributed system of honeypot sensors: Discrimination and correlative analysis of attack processes*. PhD thesis, Ecole Nationale Supérieure des Télécommunications (ENST), Paris., Jan 2006.

[153] F. Pouget and M. Dacier. Honeypot-based forensics. In *AusCERT2004, AusCERT Asia Pacific Information technology Security Conference*, May 2004.

[154] F. Pouget, M. Dacier, and V. H. Pham. Understanding threats: a prerequisite to enhance survivability of computing systems. In *Int. Infrastructure Survivability Workshop IISW'04, (25th IEEE Int. Real-Time Systems Symp. (RTSS 04)), Lisboa, Portugal, 2004.*, 2004.

[155] F. Pouget, M. Dacier, and V. H. Pham. Leurré.com: on the advantages of deploying a large scale distributed honeypot platform. In *ECCE'05, E-Crime and Computer Conference, Monaco, 2005.*, 2005.

[156] F. Pouget, M. Dacier, J. Zimmerman, A. Clark, and G. Mohay. Internet attack knowledge discovery via clusters and cliques of attack traces. *Journal of Information Assurance and Security*, 1(1):21–32, March 2006.

[157] F. Pouget and T. Holz. A pointillist approach for comparing honeypots. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2005.

[158] K. Poulsen. Wi-fi honeypots a new hacker trap. Available online at `http://www.securityfocus.com/news/552`.

[159] M. D. Preda, M. Christodorescu, S. Jha, and S.Debray. A semantic-based approach to malware detection. In *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2007.

[160] N. Provos. Honeyd sample configurations. Available online at `http://www.honeyd.org/config/wireless`.

[161] N. Provos. A virtual honeypot framework. In *Proceedings of the 12th USENIX Security Symposium*, pages 1–14, August 2004.

[162] J. Riordan, D. Zamboni, and Y. Duponchel. Building and deploying billy goat, a worm detection system. In *Proceedings of the 18th Annual FIRST Conference*, 2006.

[163] J. Rocaspana. SHELIA: A Client HoneyPot For Client-Side Attack Detection. Available online at `http://www.cs.vu.nl/~herbertb/misc/shelia/`.

[164] M. Roesch. Home page of snort. Available online at `http://www.snort.org`.

[165] S. Bhatkar, D.C. Du Varney and R. Sekar. Address obfuscation: an efficient approach to combat a broad range of memory error exploits. In *In Proc. of the 12th USENIX Security Symposium*, pages 105–120, August 2003.

[166] SANS. The sans internet storm center. Available online at `http://isc.sans.org`.

[167] SANS. Sans institute press update. `http://www.sans.org/top20/2006/press_release.pdf`, 2006.

[168] M. Schmall. *Classification and Identification of Malicious Code Based on Heuristic Techniques Utilizing Meta-languages*. PhD thesis, University of Hamburg, 2002.

[169] M. Schmall. Heuristic techniques in av solutions: An overview, 2002.

[170] B. Schneier. Organized cybercrime. Available online at `http://www.schneier.com/blog/archives/2006/09/organized_cyber.html`, Sep 2006.

[171] P. Schnoebelen. The complexity of temporal logic model checking. *Advances in Modal Logic*, 4:393–436, 2003.

[172] M. G. Schultz, E. Eskin, and E. Zadok. Data mining methods for detection of new malicious executables. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 38–49, 2001.

[173] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati. A fast automaton-based approach for detecting anomalous program behaviors. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 144–155, 2001.

[174] C. Shannon and D. Moore. The spread of the Witty worm. *Security & Privacy Magazine, IEEE*, 2(4):46–50, 2004.

[175] S. Sidiroglou, G. Giovanidis, and A. Keromytis. Using execution transactions to recover from buffer overflow attacks. *Cucs-031-04, Columbia University*, 2004.

[176] S. Sidiroglou, G. Giovanidis, and A. Keromytis. A Dynamic Mechanism for Recovering from Buffer Overflow Attacks. In *Proceedings of the 8th Information Security Conference (ISC)*, pages 1–15, Sept. 2005.

[177] P. Singh and A. Lakhotia. Static verification of worm and virus behavior in binary executables using model checking. In *Proceedings of the IEEE Information Assurance Workshop*, pages 298–300, 2003.

[178] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *In Proc. of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 45–60, 2004.

[179] S. Sinha, M. Bailey, and F. Jahanian. Shedding Light on the Configuration of Dark Addresses. In *Proceedings of the* 14^{th} *Network and Distributed System Security Sympoisum (NDSS)*, Feb. 2007.

[180] A. Slowinska and H. Bos. The age of data: pinpointing guilty bytes in polymorphic buffer overflows on heap or stack. In *23rd Annual Computer Security Applications Conference (ACSAC'07)*, Miami, FLA, December 2007.

[181] A. Smith and J. Fox. RandomNet. Available online at `http://www.citi.umich.edu/u/provos/honeyd/ch01-results/3/`, March 2003.

[182] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Welsey, Boston, 2002.

[183] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 33–42, New York, NY, USA, 2004. ACM Press.

[184] Symantec. Symantec internet security threat report. Available online at `http://www.symantec.com/enterprise/products/category.jsp?pcid=1017`, March 2007.

[185] Symantec Corporation. W32.Beagle.A@mm. Available online at `http://www.symantec.com/security_response/writeup.jsp?docid=2004-011815-3332-99`.

[186] Symantec Corporation. W32.Mydoom.A@mm. Available online at `http://www.symantec.com/security_response/writeup.jsp?docid=2004-012612-5422-99`.

[187] R. S. Systems and Networks. Technology today 2007, issue 2. Available online at `http://www.raytheon.com/technology_today/archive/2007_issue2.pdf`.

[188] P. Szor and P. Ferrie. Hunting for metamorphic. In *Virus Bulletin Conference*, pages 123–144, Abingdon, Oxfordshire, England, September 2001.

[189] The Argos Development Team. Home page of "Argos". Available online at `http://www.few.vu.nl/argos`.

[190] The Honeynet Project. Home page of "Capture-HPC". Available online at `https://projects.honeynet.org/capture-hpc`.

[191] The Honeynet Project. Home page of Sebek. Available online at `http://www.honeynet.org/tools/sebek/`.

[192] The Honeynet Project. Home page of the "Honeynet Project". Available online `http://www.honeynet.org`.

[193] The Honeynet Project. The honeynet project. Available online at `http://www.honeynet.org/misc/project.html`.

[194] The Honeynet Project. Honeynet Project, Know Your Enemy: GenII Honeynets. Available online at `http://www.honeynet.org/papers/gen2`, May 2005.

[195] The MITRE Honeyclient Project Team. Home page of "HoneyClient". Available online at `http://www.honeyclient.org`.

[196] T. Toth and C. KrÃ$\frac{1}{4}$gel. Accurate buffer overflow detection via abstract payload execution. In A. Wespi, G. Vigna, and L. Deri, editors, *Recent Advances in Intrusion Detection, 5th International Symposium*, volume 2516 of *Lecture Notes in Computer Science*. Springer, October 2002.

[197] J. Tucek, S. Lu, C. Luang, S. Xanthos, Y. Zhou, J. Newsome, D. Brunmley, and D. Song. Sweeper:a light-weight end-to-end system for defending against fast worms. In *Proceedings of Eurosys 2007*, Lisbon, Portugal, April 2007.

[198] U. Shankar, K. Talwar, J. S. Foster, and D. Wagner. Detecting format string vulnerabilities with type qualifiers. In *In Proc. of the 10th USENIX Security Symposium*, pages 201–216, August 2001.

[199] UploadMalware. Home page of "UploadMalware". Available online at `http://www.uploadmalware.com`.

[200] V. Kiriansky, D. Bruening and S. Amarasinghe. Secure execution via program shepherding. In *In Proc. of the 11th USENIX Security Symposium*, 2002.

[201] N. Vanderavero, X. Brouckaert, O. Bonaventure, and B. Charlier. The honeytank : a scalable approach to collect malicious internet traffic. In *Proceedings of the International Infrastructure Survivability Workshop (IISW'04)*, Dec. 2004.

[202] F. Veldman. Heuristic anti-virus technology. In *Proceedings of the International Virus Protection and Information Security Council*, 1994.

[203] VMware, Inc. Home page of "VMware". Available online at `http://www.vmware.com`.

[204] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP), Brighton, UK*, Oct. 2005.

[205] D. Wagner and D. Dean. Intrusion detection via static analysis. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 156, Washington, DC, USA, 2001. IEEE Computer Society.

[206] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. *SIGCOMM Comput. Commun. Rev.*, 34(4):193–204, 2004.

[207] J.-H. Wang, P. S. Deng, Y.-S. Fan, L.-J. Jaw, and Y.-C. Liu. Virus detection using data mining techniques. In *Proceedings of IEEE on Security Technology*, pages 71–76, 2003.

[208] K. Wang. Using honeyclients to Detect New Attacks. Available online at `http://www.synacklabs.net/honeyclient/Wang-Honeyclient-ToorCon2005.pdf`.

[209] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection(RAID 2005)*, 2005.

[210] X. Wang, Z. Li, J. Xu, M. K. Reiter, C. Kil, and J. Y. Choi. Packet vaccine: black-box exploit detection and signature generatio n. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 37–46, New York, NY, USA, 2006. ACM Press.

[211] X. Wang, C.-C. Pan, P. Liu, and S. Zh. Sigfree: A signature-free buffer overflow attack blocker. In *Proceedings of 15th USENIX Security Symposium*, 2006.

[212] Y. 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 *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, pages 39–58, Feb. 2006.

[213] 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 *Proc. Network and Distributed System Security (NDSS)*, San Diego, CA, February 2006.

[214] M. Webster. Algebraic specification of computer viruses and their environments. In *Selected Papers from the First Conference on Algebra and Coalgebra in Computer Science Young Researchers Workshop (CALCO-jnr 2005), University of Wales Swansea Computer Science Report Series (CSR 18-2005)*, pages 99–113, 2005.

[215] M. Webster and G. Malcolm. Detection of metamorphic computer viruses using algebraic specification. *Journal in Computer Virology*, 2(3):149–161, 2006.

[216] T. Werner. Honeytrap. Available online at `http://honeytrap.sf.net`.

[217] M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proc. of ACSAC Security Conference*, Las Vegas, Nevada, 2002.

[218] Wired.com. Cybercrime is getting organized. Available online at `http://www.wired.com/news/wireservice/0,71793-0.html`, Sep 2006.

[219] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In *Proceedings of the Recent Advance in Intrusion Detection (RAID) Conference 2004*, September 2004.

[220] M. Zalewski. Home page of "p0f". Available online at `http://lcamtuf.coredump.cx/p0f.shtml`.

[221] M. Zalewski and W. Stearns. Passive OS Fingerprinting Tool. Available online at `http://lcamtuf.coredump.cx/p0f.shtml`.

[222] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou. Collecting autonomous spreading malware using high-interaction honeypots. *Proceedings of ICICS*, 7, 2007.

[223] D. A. D. Zovi and S. Macaulay. Karma wireless client security assessment tools. Available online at `http://www.theta44.org/karma/`.

[224] U. Zurutuza Ortega. *Data Mining Approaches for Analysis of Worm Activity Toward Automatic Signature Generation*. PhD thesis, Mondragon University, November 2007.

[225] R. Zwienenberg. Heuristics scanners: Artificial intelligence? In *Proceedings of the Virus Bulletin Conference*, pages 203–210, 1994.