SEVENTH FRAMEWORK PROGRAMME

Information & Communication Technologies
Trustworthy ICT

NETWORK OF EXCELLENCE

A European Network of Excellence in Managing Threats and Vulnerabilities in the Future Internet: *Europe for the World* [†]

Deliverable D5.1: Survey of Research and Data Collection Initiatives in Malware and Fraud

Abstract: This deliverable presents a survey of research works and data collection initiatives in the areas of malware detection and analysis and on-line fraud.

Contractual Date of Delivery	May 2011
Actual Date of Delivery	June 2011
Deliverable Dissemination Level	Public
Editor	Paolo Milani Comparetti
Contributors	TU WIEN EURECOM POLIMI FORTH

[†] The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257007.

The *SysSec* consortium consists of:

FORTH-ICS	Coordinator	Greece
Politecnico Di Milano	Principal Contractor	Italy
Vrije Universiteit Amsterdam	Principal Contractor	The Netherlands
Institut Eurécom	Principal Contractor	France
IPP-BAS	Principal Contractor	Bulgaria
Technical University of Vienna	Principal Contractor	Austria
Chalmers University	Principal Contractor	Sweden
TUBITAK-UEKAE	Principal Contractor	Turkey

Contents

List of Figures

1 Introduction

This deliverable provides an overview of existing research in the areas of malware and fraud. However, rather than attempting to cover all relevant research on malicious activity on the internet (which would be a daunting task), this document is narrowly focused on discussing approaches to data collection, data sharing, and data-driven research in malware and fraud. In the context of systems security, an experimental approach based on collecting and analyzing real-world data is a relatively recent development. This is in part because of the difficulty of obtaining meaningful, representative datasets on malicious activity and because of the legal and ethical obstacles to sharing data. Despite these difficulties, over the last decade a significant body of work has accumulated that raises to the challenge of collecting and analyzing real data; Testing and evaluating detection and analysis techniques against a signifcant and representative selection of real threats has become a "best practice" for systems security research.

In recent years, several *SysSec* partners have contributed to this shift towards empirical, data-driven systems security research with their work within the WOMBAT EU project. In addition to performing research and developing novel techniques for collecting and analyzing data on malware and fraud, the WOMBAT project proposed and implemented a practical approach for sharing data between security researchers. At the conclusion of the WOMBAT project, the partners organized the first BADGERS workshop [44], which solicited contributions from security researchers to discuss datasets, data collection, data sharing and data analysis for security research, not only from a technical perspective but also from a legal point of view. The success of this workshop emphasized the complexity of these problems and the interest for these topics in the systems security community.

The topic of data-driven security research has also been discussed within the *SysSec* project, and specifically within the Malware and Fraud Working Group. While several *SysSec* partners are continuing, in the footsteps of WOMBAT, to work on advancing data-driven systems security research, the *SysSec* Malware and Fraud Working Group has also been discussing the limits of this approach. For our medium-term research roadmap, we are interested on the one hand on understanding and improving upon the limitations and "blind spots" of the security data that is currently available to researchers. On the other hand, we would like to understand how to overcome the more fundamental limitations of data-driven security research when facing hard-to-observe phenomena, such as targeted attacks.

This deliverable provides an overview of the state of the art in research and data collection activities in malware and fraud. Chapter 2 provides related work on honeypots, including traditional server honeypots as well as more recent client-side honeypot technology. Chapter 3 provides an overview of malware analysis platforms, and analysis techniques that can be employed on the analysis artefacts they produce. Chapter 4 discusses research that aims to understand the underground economy of internet crime by measuring online fraud from technical and economics perspectives. Finally, Chapter 5 discusses research on detecting network worms and botnets by observing the network traffic they generate.

2 Honeypots

Honeypots, as they were proposed originally, are instrumented systems that monitor an unused portion of IP address space with the sole purpose of trapping attackers. They have proven themselves to be a valuable tool for capturing and analyzing both known and unknown attacks. Security researchers and large anti-virus companies use them to collect malware samples and acquire information of the nature and purpose of cyberattacks. A number of design approaches have been proposed. Most fundamental trade-offs among these designs have to deal with the interactivity of a honeypot, its instrumentation level and finally its detectability. The purpose of this work is to provide a detailed review of existing honeypot systems and technologies, compare their characteristics and present the most outstanding papers in the area of honeypots.

2.1 Introduction

Worms, viruses, trojans, keyloggers, malware in general, are a constant threat to the Internet. Day by day, more instances and variants of malware appear, discouraging people from using Internet at its full potential. To combat malicious software, several approaches have been proposed, each one followed by hundreds of implementations, products and research papers. Examples are firewalls, intrusion detection systems, anti-virus products and honeypots. Honeypots are the most recent development in the area of network defense. Honeypots are a valuable tool for gathering and analyzing information considering cyber-attacks. We cannot argue that honeypot technology can solely defeat all existing and future threats, but it is widely used as a large information source for malware activities. In conjuction with other approaches, honeypots can greatly help towards effective large-scale defense mechanisms.

A formal definition of a honeypot is a "trap set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems"[1]. Practically, honeypots are computer systems set up to lure attackers. They 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. By default, all traffic destined to honeypots is malicious or unauthorized as it shouldn't exist in the first place. Honeypots can also assume other forms, like files, database records, IP addresses or e-mails. In this report, we will focus on honeypots that listen to unused IP addresses.

Several honeypot designs have been proposed. The two main axes on which a honeypot is designed is the level of interactivity with the attackers and the side we want to protect. Concerning the level of interactivity, honeypots can either do simple service emulation (low-interaction), more advanced emulation (medium-interaction) or run real services (high-interaction). As far as the second axis is concerned, we can categorize honeypots as server-side and client-side. Most honeypots protect the server side but client-side honeypots follow a different approach. Instead of waiting to be attacked, they search for attackers. We further explore design requirements and tradeoffs in Section 2.2.

2.2 Comparison framework

To classify the surveyed systems we use the following properties and features as axes of a comparison framework.

- **Realism:** The role of honeypots is to interact with attackers and convince them that they are attacking a real system. A honeypot should be as realistic as possible both in terms of what services it runs and what responses it sends back to attackers. Using simple emulation scripts or unrealistic profiles of running services (e.g. emulate all possible services in a single honeypot) can result in getting the honeypots into the attackers' blacklist.

  There are three levels of interaction that determine how realistic a honeypot can be. Low-interaction honeypots provide the minimum level of interaction. They run scripts that emulate several services. Emulation scripts are not as accurate as the real service but in some cases they have proved to be enough to attract automated attacking tools. Furthermore, low-interaction honeypots run in a raw fashion, that is they emulate the network stack. Medium-interaction honeypots also use emulation scripts but are more resistant to fingerprinting

[1] http://en.wikipedia.org/wiki/Honeypot_%28computing%29

attacks as they do not handle the network stack on their own. High-interaction honeypots provide the maximum level of realism as they run real services.

- **Scalability:** Honeypots are often used to monitor portions of unused IP address space and respond to attackers. The size of the monitored IP address space may span from a few to several thousands addresses. The latter case could easily result in hundreds or even thousands of connection requests per second. In cases where the size of the monitored address space is not fixed, it is important for a honeypot system to be able to scale and continue to respond in a timely fashion to incoming requests as more addresses are added to the monitored address space.

- **Instrumentation level:** In its simplest form, a honeypot can run services and receive attacks without any instrumentation. However, taking that approach can tell us who attacked the honeypot and what service she targeted but not any further information for the vulnerability that was exploited. Instrumentation enables honeypots to have full control over the running services, detect when an attack is taking place (and even stop it before it infects the honeypot) and gather information about the disclosed vulnerability which can be used to generate protective filters.

- **Exposure:** A major concern with honeypots is that they may get infected as they are attack targets. While for low- and medium-interaction honeypots this is not the case, high-interactions are likely to get compromised and try to infect other hosts. The level of exposure is a significant design property. Several approaches run the services inside virtual machines and once they are infected, the virtual machine reboots from a clean state. Other approaches control the infection through specialized gateways at the edge of the honeypot subnet that either block attack traffic, perform traffic shaping or reflect the traffic to internal hosts. Finally, another solution is to fully instrument the services and prevent the infection at the moment the vulnerable service is getting exploited.

- **Direction:** Originally, honeypots were designed to lure and analyze attacks directed to servers. However, over the recent years, client-driven attacks have been extremely popular and effective. In order to cope with this problem, client-side honeypots follow a different approach. While server-side honeypots wait passively to get attacked, client-side crawl various communication channels, such as the Web or IRC, to locate malware sources and botnets.

- **Ease of installation and maintenance:** The installation and configuration of a honeypot is a manual procedure that requires significant human effort. A honeypot system should relieve administrators from major overheads and run with little maintenance overhead. As there are few solutions for auto-configuration and auto-recovery, a honeypot must minimize the downtime and effort required to keep all its services running and ensure it is not compromised.

- **Detectability:** As any defense system, honeypots must be undetectable. For systems like firewalls or intrusion detection systems, which do not interact with attackers, detectability is rarely an issue. However, as honeypots respond to malicious activities they are vulnerable to be detected and thus blacklisted. Earlier implementations of honeypot systems were trivially discovered using TCP/IP fingerprinting techniques. Virtual machines are also identifiable using a number of techniques, like device probing or timing attacks. Apart from the technical aspect, honeypot configuration is also a point that can reveal the location of a honeypot. Having thousands of ports open on a single host or running unrealistic services (e.g. honeypots running Unix SSH daemon while production network is a Windows dominated environment) can raise suspicions to attackers.

2.3 Related work

In this Section we present the related work in the area of honeypots and we structure it as follows. First, we present server-side honeypot systems based on their interactivity level; low-interaction (Section 2.3.1), medium-interaction (Section 2.3.2) and high-interaction (Section 2.3.3). Second, we present an overview of client-side honeypots in Section 2.3.4. Opposed to the the previous three categories, client-side honeypots present different properties and form a special category. Our overview continues with a presentation of most well-known and widely used architectures that are based on the honeypot tools and systems we describe in Sections 2.3.1, 2.3.2, 2.3.3 and 2.3.4. Finally, in Section 2.3.6 we present research papers that study honeypot performance and detectability issues.

2.3.1 Low-interaction honeypots

A low-interaction honeypot tries to emulate real services and features of operating systems, usually through specialized components. The emulation components are designed to mimic real software but their scope is limited. The main advantage of low-interaction honeypots is that they are very lightweight processes and their emulated services are unlike to be infected as they are usually scripts or responders that imitate real responses. Their

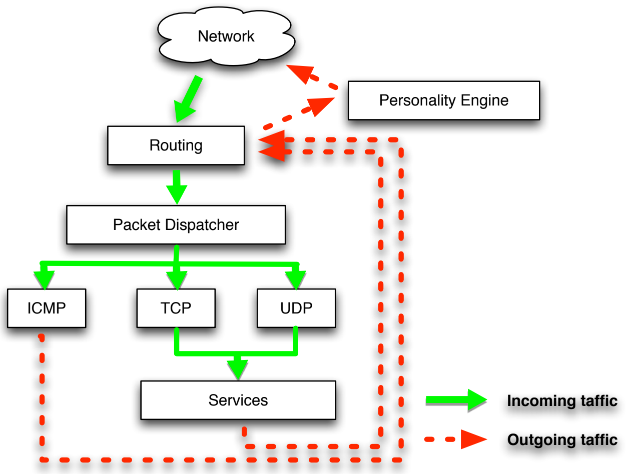


Figure 2.1: Architecture of honeyd. Green (solid) lines indicate how incoming traffic is handled. The routing process sends the packets to the packet dispatcher which decides on which handler to deliver the packet to based on the protocol. Handlers send the traffic to specific services based on the destination port. The responses from the services (red dotted lines) are sent back to the routing process, which consults the personality engine before injecting them to the network.

main disadvantage is that emulation is inaccurate and can only detect previously known attacks.

2.3.1.1 Honeyd

The most popular low-interaction honeypot is honeyd [144] . 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 components that emulate latency and loss characteristics of the network. A 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 in the IP address 10.1.0.2, a machine running Windows XP Service Pack 2 exists. Additionally, this virtual host will have port 80 open and the port will be handled by the "web.sh" script. All emulation is done through scripts, usually written in the Perl language. A very interesting property of honeyd is its ability to imitate network stacks of various operating systems. Honeyd does not bind to any sockets; instead it performs network stack emulation, thus it is highly scalable and can listen to arbitrarily large address spaces. Network stack emulation is more advanced than simply maintaining state for each connection. As the implementation of network protocols do not confront to standards, each operating system's network stack fills certain fields in their own way, for example the IP identification number or TCP timestamps. Honeyd maintains a database of network stack characteristics, originally created by p0f[19] tool. These profiles are called *personalities*. When it needs to respond on behalf of a virtual host that "runs" Windows XP SP2, the database is advised and packets are formed in such a way to resemble those sent by a normal SP2 operating system. This property makes the virtual honeypot resilient to remote OS detection scans, like the ones performed by nmap[18] tool. 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.

2.3.1.2 Honeytrap

Honeytrap[7], developed by TillmanWerner, 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 to 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. Thus, Honeytrap view on the network attacks is thus not uniform, but heavily depends on their structure and their complexity.

2.3.1.3 LaBrea

LaBrea[8] is a program that listens to an 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.

2.3.2 Medium-interaction honeypots

Medium-interaction honeypots also emulate services but, unlike low-interaction honeypots, they do not manage network stacks and protocols themselves. Instead, they bind to sockets and let the operating system do the connection management. In contrast to 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.

2.3.2.1 Nepenthes

The nepenthes platform[16] is a system that was designed to automatically collect malware[2]. 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

[2]Historically, nepenthes is the evolution of mwcollect platform

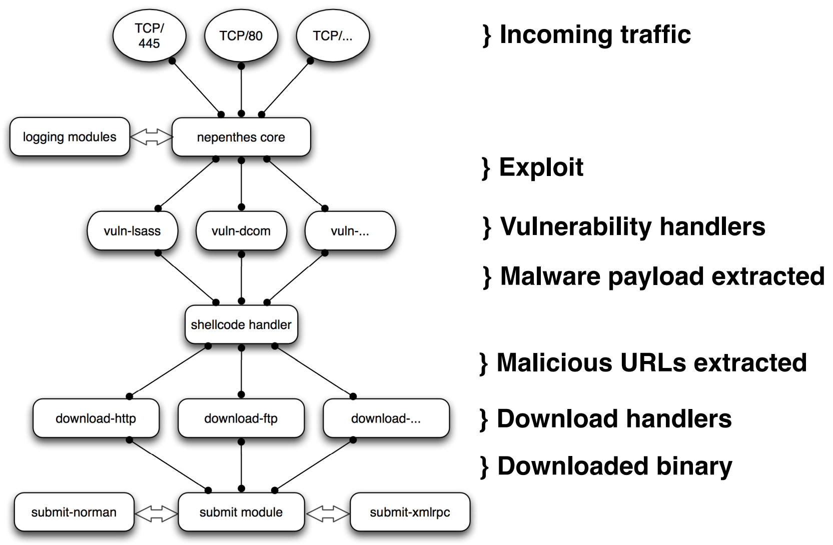


Figure 2.2: The architecture of Nepenthes. Incoming traffic is handed over to the appropriate vulnerability handler based on the destination port. Vulnerability handlers extract the expoit and invoke the shellcode manager. The shellcode manager emulates the shellcode and extracts the URLs it contains. The URLs are given to download handlers who actually download the malicious binary. Downloaded binaries are either stored locally or/and submitted to a central repository.

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.

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 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.

2.3.2.2 Multipot

Multipot[14] 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[15] and Beagle[3]). 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.

2.3.2.3 Billy Goat

Billy Goat[101] is a 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 solution for the emulation of SMB protocols, that consists in taking advantage of a hardened version of the open-source implementation of the protocol[22]. Such an implementation is derived from the work done by Overton[136], 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 [180].

2.3.3 High-interaction Honeypot

High-interaction honeypots, unlike the two previous categories, do not emulate services. On the contrary, they run 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[34], Qemu[21] and Xen[59]. 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 three high-interaction honeypots, both based on virtual machine technology.

2.3.3.1 Minos

Minos[74] is a microarchitecture that implements Biba's low water-mark integrity policy[66] on individual words of data. Minos stops attacks that corrupt control data to hijack program control low. Control data is any data that is loaded into the program counter on control-low 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 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. Minos's threat model thus assumes that the kernel is trusted, and cannot itself be compromised. 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 is going to be executed, then an attack is taking place.

Minos was emulated on the 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.

2.3.3.2 Argos

Argos[142] 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 is used as jump targets, function addresses or instructions. To identify such activities, Argos performs dynamic taint analysis[133] (memory tainting). Memory tainting is the process where unsafe data that resides in the main memory or the registers are tagged. All data coming from the network is marked as tainted because by default they are considered as unsafe. Tainted data is 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 enters the Argos emulator, it is recorded in a network trace. As Argos has control of all operations that happen in the guest OS[3], it can detect whenever tainted data is 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

[3]In virtual machine terminology, guest OS is the operating system running inside the virtual machine while host OS is the operating system that runs the virtual machine software

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 is 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 it is able to detect without false positives that an automated attack is taking place, regardless of the application under attack or the attacks 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[21] 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.

2.3.3.3 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 [96]. 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[59] 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 is 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[11], which measures time for well-known system calls.

2.3.3.4 HoneySpot

HoneySpot is an architecture designed by the Spanish Honeynet Projet (SHP) that aims at monitoring the attacker's activities in wireless networks [145]. 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 traffic 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.

2.3.4 Client-side honeypots

Recently, we have observed exploits that target client applications and especially web browsers. The WMF and JPEG vulnerabilities ([13],[12]) have shown how the Internet Explorer browser can be compromised and execute arbitrary code on the victim's side. Instead of waiting passively for the attackers to contact them, as we have seen so far, client-side honeypots try to spot locations where malicious content is hosted. Although they are not passive systems, they are still characterized as honeypots as they are non-production systems. Client-side honeypots try to cover the gap of classic detection techniques. According to [32], only 1.5% of IDS signatures are based on client-side attacks, although the number of client-based vulnerabilities increases over time. In this Section strider honeymonkeys and honeyclient, the most popular client-side honeypots, are presented.

2.3.4.1 Strider HoneyMonkeys

Strider Honeymonkeys system[169] 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.

2.3.4.2 HoneyClient

HoneyClient[6] 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.

2.3.4.3 Capture

Capture[4] is a high-interaction client honeypot that tries to find malicious servers on the network following a similar approach with HoneyMonkeys. It consists of two main components, Capture Server and Capture Client. The primary purpose of the Capture Server is to control numerous Capture clients to interact with web servers. It allows to start and stop clients, instruct clients to interact with a web server retrieving a specified URI, and aggregating the classifications of the Capture clients regards the web server they have interacted with. The server provides this functionality in a scripting fashion. The URIs are distributed to the client in a round robin fashion.

Capture clients accept the commands of the server so as to know when to start and stop themselves and which URI to visit. A Capture client runs inside a virtual machine and interacts with a web server, monitoring its state for changes on the file system, registry, and processes that are running. Since some events occur during normal operation (e.g. writing files to the web browser cache), exclusion lists allow to ignore certain type of events. If changes are detected that are not part of the exclusion list, the client makes a malicious classification of the web server and sends this information to the Capture server. Since the state of the Capture client has been changed, the Capture client resets its state to a clean state before it retrieves new instructions from the Capture server. In case no state changes are detected, the Capture client retrieves new instructions from the Capture server without resetting its state. Capture allows to automatically collect network traces and downloaded malware files when a malicious server is encountered.

2.3.4.4 Shelia

Shelia[24] is a high-interaction client-side honeypot that tries to locate malware sources by inspecting the mail of the user and more specifically her spam messages. Shelia is implemented for Windows platforms and currently works with the Outlook Express mailer. Spam messages can contain all kind of information. They can include various types of attachments, URLs for malicious websites or even HTML code that can lead to infection.

The architecture of Shelia includes two main components. The first one is the client emulator which is responsible for identifying all the attachments and URLs received by email and to invoke the proper application to handle these attachments/URLs. The second one is the attack detector. The attack detector has a process monitor engine that monitors the client's actions and applies the containment strategy if required. The detector also determines if an action taken by a client is illegal. All attack information is logged by the attack log engine. Shelia monitors the processes and generates alerts when the process attempts to execute an invalid operation (i.e., execute a call to change the registry, create files, or attempt specific network operations) from a memory area that is not supposed to be executable code. The process monitoring is performed by hooking Win32 API calls, such as CreateProcessA, CreateThread, LoadLibraryA etc. The containment strategy followed by Shelia may allow the attack to run until it downloads the malware, which is then captured and stored in a specific directory. However, the downloaded malware is never executed.

2.3.5 Honeypot Architectures

2.3.5.1 Collapsar

Collapsar[102] 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 2.3. 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 redirector is implemented as a virtual machine running an extended version of User-Mode Linux[31] (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 redirector 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 attack other hosts on the Internet. To prevent such malicious activities, introduces three assurance models: logging, tarpiting and correlation. The logging module is embedded in the honeypots guest OS as well as log storage in the physical machines host OS in order to be invisible to the attacker. The tarpiting module throttles outgoing traffic from honeypots by limiting the transmission rate and also scrutinizes outgoing traffic based on known attack signatures. Snort-inline performs this task. Snort-inline is a modified version of Snort[25], a very popular intrusion detection system. While Snort is a system that passively monitors traffic, Snort-inline intercepts 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.

2.3.5.2 NoAH

The NoAH project[17] follows an approach similar to the Collapsar architecture. Since centralized honeypot farms provide a limited view of Internet 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[79]. 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.

2.3.5.3 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[168] 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 honeypots 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 honeypots 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.

2.3.5.4 Leurre.com

Leurre.com[9] is a distributed honeypot environment that operates a broad network of honeypots covering around 30 countries. Honeypots run a modified version of honeyd and emulate three different operating systems; two from the Windows family (98 and NT server) and Redhat 7.3. Traffic and security logs are retrieved daily and stored into a centralized database. Apart from logs, raw traffic is also analyzed, mainly to derive information about attackers and specifically IP geographical location, DNS names, OS fingerprinting and TCP stream analysis. 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. Participants in the Leurre.com project need to deploy a platform and in return they are granted access to the centralized database.

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.

2.3.5.5 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[37] 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 three operations: data capture, data collection and data analysis. 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[23] 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. The attacker must not be able to detect 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 [80]. 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 vectors 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 published several "Know your enemy" white-papers available at project's website[4]. Honeynet farms were originally used to capture and analyze manual attacks. Recently, they have been used to track phishing attempts and botnets.

2.3.5.6 A hybrid honeypot infrastructure

Provos et al. in [55] 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 estab-

[4]`http://www.honeynet.org`

lishes 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.

2.3.5.7 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[46] 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 a 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 that are heavily instrumented so as to detect attacks. Two types of shadow honeypots can be identified: loosely-coupled and tightly-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 [155]. 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%.

2.3.5.8 Vigilante

Vigilante is an infrastructure that aims for 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 the lack of information about software vulnerabilities. End hosts act as honeypots; they run instrumented versions 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 is marked as dirty and if dirty data is going to be executed or used as critical arguments for a function, a signal for exploit is raised. Unlike approaches described in Section 2.3.3, 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 is 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 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 vulnerability. 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.

2.3.5.9 iSink

The iSink architecture[176] 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[109]. 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 any rate 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.

2.3.5.10 Internet Motion Sensor

The Internet Motion Sensor (IMS)[51] is a Internet monitoring system covering 28 unused IP blocks, ranging from /25 to /8. 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 the data of the actual attack can be seen. The lightweight responder 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 in all cases, it is enough for many cases like the Blaster worm that did not wait for any application responses. To avoid the 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 paper provides a number of traffic statistics over a 7-day period and uses the Blaster worm as an example.

2.3.5.11 Honeystat

HoneyStat[77] 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[73] or Windows logs. Each outgoing TCP SYN or UDP traffic is a network event. Disk events are generated by activities that try 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 breakins 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.

2.3.5.12 HoneyTank

The HoneyTank project[166] 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 occurrences of the attacks were measured (20 attacks were detected). Authors also compared HoneyTank with Darknet[76]. 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.

2.3.6 Research papers

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

2.3.6.1 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[122] 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[29], and an algorithm proposed by authors, the Region Analysis algorithm. Finally, from the simplified state machine a honeyd script is generated. 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 may, or may not, have 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.

2.3.6.2 A Pointillist Approach for Comparing Honeypots

Pouget and Holz in [143] 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.

2.3.6.3 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 [157] 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[158] (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 it is easy to discover 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 a variation in the threat landscape. Ports that are frequently attacked in one month, may be 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 to use the authors' approach.

2.3.6.4 Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic

M. Bailey et al. in [53] 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 2.3.5.10. 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 the 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.

The authors evaluated three methods of source-based filtering previously reported in [138]: 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 were 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. The author's approach examines 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.

2.3.6.5 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 [2], 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. Similar detection techniques can be applied to other virtual machines or emulators such as Qemu and Bochs.

Detection techniques against Honeynets, and especially Sebek, are also described in [81]. 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.

2.4 Comparison

In this Section we make a comparison of the different honeypot types. We have selected the tools that are widely deployed. The comparison axes have been extensively described in Section 2.2. A summary of the comparison is shown in Table 2.1.

All low-interaction honeypots present a low level of realism as they emulate services using scripts, which do not implement all protocol semantics. Although they are highly scalable, they are also highly detectable as the incomplete implementation of their service emulation can reveal their role. The exception is LaBrea but we should bear in mind that the original purpose of LaBrea is to stale connections by attackers. Low-interaction honeypots do not suffer from the danger of being compromised as they perform emulation and not real services that are vulnerable to attack vectors. We have not marked the exposure level to zero as all machines connected to the Internet can be compromised or the emulation software may also be vulnerable (e.g. their packet processing routines can be overflown). To the best of our knowledge no vulnerabilities have been reported up to date for all three low-interaction honeypots shown in Table 2.2. Concerning the ease of installation, Honeyd presents a medium degree as the initial configuration overhead is significant. Automated tools however help towards this direction. The other two tools, LaBrea and Honeytrap, have minimal configuration settings to be set up.

Medium-interaction honeypots present a better level of realism as they are not vulnerable to network fingerprinting and use more advanced emulation services. Billy Goat in fact does not use emulation but a hardened version of the SMB protocol. However, we include it in the category of medium-interaction honeypots as it does not run any other service and its basic operation is similar to the one of Honeyd. Nepenthes, on the other side, uses emulation scripts for various services which makes it vulnerable to fingerprinting. The scalability of both tools is medium as they can bind only to few addresses per physical machine (recall that they do not perform raw operation to monitor and handle traffic and that separates them from low-interaction honeypots).

Argos and Minos share the same basic principle so most of the characteristics match. Both tools present low scalability as their processing and memory overhead allows to run only a few instances (no more than 3-4) per physical machine. High-interaction honeypots present the maximum level of realism as they run full services in their native operating system. However, the high level of instrumentation eliminates most of the exposure risks. Both approaches detect the attack before it launches so no malicious code will run inside the virtual machine to perform the detection and the node compromise. Honeynets follow a different approach. By not performing instrumentation, it becomes vulnerable to being compromised. Their scalability is poor as each physical machine can monitor one or few IP addresses (per interface). Honeynets are also vulnerable to detection, mainly due to their Sebek module that monitors the kernel activities. The documentation and wide community of Honeynets makes the installation of a honeynet an easy process.

All implementations of client-side honeypots share some common characteristics. First of all, due to their nature, they have low chances to get detected as they behave as normal browsers that surf the World Wide Web. The chance to blacklist the IP address range of a client-side honeypot is also minimal as it is hard to distinguish them from normal crawlers and bots or they might use proxies. Second, as all approaches run real browsers in their native environment, they provide the highest level of realism. Furthermore, they have plugins enabled (e.g. Flash and Java) so they can detect third-parties vulnerabilities. A recent example was the vulnerability in Flash applications[1]. A client-side honeypot is able to detect all these pages that try to infect users through a Flash applet. The instrumentation level differs per approach. Finally, they all follow the same instrumentation pattern, monitoring the filesystem and registry for changes. The scalability of client-side honeypots varies. As client-side honeypots act like crawlers, they are demanding in terms of bandwidth and physical machines. As the space of web pages is enormous, tens of machines and several hundreds of Mbps are needed to crawl the web at a pace of thousands of pages per second. The extra security components needed by HoneyClient makes it less scalable than the other two. Concerning the ease of installation, Capture and HoneyClient have an extensive documentation that helps the process. As Strider Honeymonkeys is a proprietary project, we could not have an idea of how easy it is to install it. We intentionally left the Shelia tool out as it is an offline processing tool for inspecting spam e-mail for malicious URLs and attachments.

Name	Side	Realism	Scalability	Instrumentation	Exposure	Installation ease	Detectability
Honeyd	Server	Low	High	None	Low	Medium	High
Labrea	Server	Low	High	None	Low	High	Low
Honeytrap	Server	Low	High	None	Low	High	High
Billy Goat	Server	Medium	Medium	Low	Low	High	Low
Nepenthes	Server	Medium	Medium	None	Low	High	High
Argos	Server	High	Low	Full	Low	Medium	Low
Minos	Server	High	Low	Full	Low	n/a	Low
Honeynets	Server	High	Low	Low	High	High	High
Strider HoneyMonkeys	Client	High	Medium	High	Low	n/a	Low
HoneyClient	Client	High	Low	Medium	Low	High	Low
Capture	Client	High	Medium	Medium	Low	High	Low
Shelia	Client	n/a	Low	Medium	Low	Medium	n/a

Table 2.1: A comparison overview of server-side and client-side honeypots

2.5 Summary and concluding remarks

Honeypot technology is a powerful addition to the set of existing defense mechanisms. Honeypots provide an abundance of information in terms of attack context, malware executables, triggered exploits and attack behavior. An advantage of honeypots is that they are even capable of capturing previously unknown attacks (0-day). Their main disadvantage is that they are still demanding in terms of resources, provided we want a good level of attack detection and containment, and difficult to configure and maintain.

In this review, we presented all four types of honeypots; most well-known low-, medium-, high- and client-side honeypots were described and their advantages and drawbacks were highlighted. Based on honeypots, a number of architectures have been proposed. Collapsar, Potemkin, Leurre.com, honeynets, Vigilante, shadow honeypots are only some architecture examples. We cannot argue that a single architecture is sufficient to address the problem of cyberattacks, however a combination of them will yield a more complete and coherent solution. Apart from architectures, a number of research works were presented. These works focused on comparing the efficiency of honeypots, their detectability against various types of discovery attacks and finally their configurability. While efficiency, performance and configurability issues are partially addressed, the detectability of honeypots still remains an open research issue.

Concluding, honeypot technology is a very interesting research with lots of research issues remaining open. Our hands-on experience with honeypots has shown that they are able to catch more attack information than we can handle and the lack of automated tasks have made them a rigid solution. Over the last few years, however, security researchers, ISPs and CERTs deploy honeypots in their effort to understand and prevent cyberattacks. This increasing trend reveals that although honeypot technology is not chronologically recent, it plays an important role and attracts many experts. We expect to see more honeypot-related works in the near future.

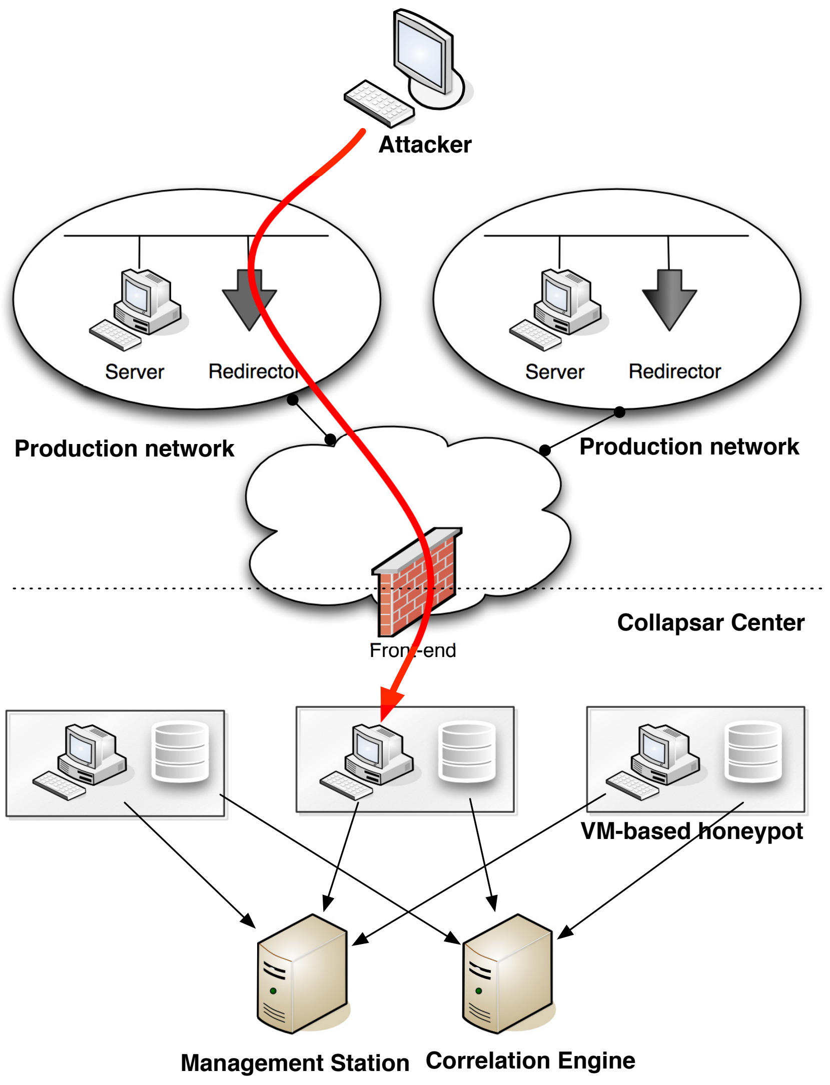


Figure 2.3: Architecture of Collapsar

3 Malware

The term Malware (**Mal**icious soft**ware**) comprises a relatively broad spectrum of software. Per definition it refers to a software designed to secretly access a computer system without the owner's informed consent. The expression itself is a general term and describes a wide variety of program code.

Honeypots, as they were described in the previous section, strictly aim at gathering executables for further analysis. Therefore, these mechanisms are one of the first entry points when collection data about malicious software. As soon as a collected malware sample is being analyzed, however, the honeypot-domain is left. How the result of such an analysis is used, strongly depends on its purpose. Antivirus companies, for instance, usually follow a different agenda than researchers.

In this Section, a description of existing methods and implementations to share data and analysis results between different frameworks and tools is given.

3.1 Introduction

The Internet has become a very important aspect of our lives. Undoubtedly, Internet applications are the most dominant way to provide access to online services today. Every day, millions of users purchase items, transfer money, retrieve information and communicate over the Internet. Although it is convenient for many users because it provides anytime, anywhere access to information and services, at the same time, it has also become a prime target for miscreants who attack unsuspecting Internet users with the aim of making an easy profit. The last years have shown a significant increase in the number of Internet-based attacks, highlighting the importance of techniques and tools for increasing the security of the Internet. For example, online banking web sites all over the world are frequent targets of scamming attempts and there has also been extensive press coverage of recent security incidences involving the loss of sensitive credit card information belonging to millions of customers. Most of these attacks are enabled by some sort of malware deployed at the enduser. In almost all cases without the user's knowledge.

Defensive techniques to protect machines from a malware infection often rely on signature-based approaches. Virus scanners, for example, are only as effective as the underlying methodology to create the signatures for recently encountered malware. It is quite obvious that the quality of such a scanner highly depends on

a) the amount, coverage and speed with which new malware samples are gathered. So called *0-day* malware, which entitles previously unseen exploit code, must be analyzed as soon as possible, to be able to distribute the corresponding signatures to the users

b) the quality of the analysis result. The better a new sample can be assessed automatically, the faster an update for virus definition databases can be shipped.

When talking about Virus scanners, this usually means that a company is dealing with the process of gathering samples, analyzing them and distributing updates to their customers. However, the same problem applies to researches and the academia as well. Ideally, a researcher would want to operate on a very rich dataset with detailed analysis results, no time delay on occuring 0-days and real-time processing of new samples.

In reality, researchers try to focus on a very specific problem like enhancing honeypots, finding new ways to analyze samples or clustering the results for a better view. As a result, they often lack the resources to properly tackle related but equally important tasks. The logical solution is, to share data between interested parties. In addition to the technical questions, there are organizational means that need to be in place to allow people to cooperate.

In the following, some of the most important means to analyze malware binaries and how they produce, share and evaluate their data will be discussed.

3.2 Sandboxing

Studies of binary analysis framework have shown, that a static code analysis is always limited in its overall efficiency and effectiveness. Therefore, dynamic code analysis frameworks that are based on sandboxing techniques are the tools of choice when it comes to automatically assess a binary's behaviour. Sandboxing refers to executing arbitrary code in a closed, virtual environment and monitoring its behaviour. Implementations based on this technique can produce very good results and are not prone to various obfuscation techniques and self modifying code.

One possible limitation of this approach, however, is that malware may attempt to evade analysis by detecting the sandbox environment and refusing to perform its malicious functionality [119, 151, 146, 137, 86, 87, 63]. Therefore, this approach may have to be complemented with techniques aimed at detecting and circumventing sandbox evasion attempts [70, 58, 103, 104].

3.2.1 Anubis

Anubis [62, 64], which is an implementation of such an approach based on the virtualization environment Qemu [21], provides insights into common malware behaviors. Qemu is a fast PC emulator that properly handles self-modifying code. To achieve high execution speed, Qemu employs an emulation technique called dynamic translation. Dynamic translation works in terms of basic blocks, where a basic block is a sequence of one or more instructions that ends with a jump instruction or an instruction modifying the static CPU state in a way that cannot be deduced at translation time. The idea is to first translate a basic block, then execute it, and finally translate the next basic block (if a translation of this block is not already available). The reason is that it is more efficient to translate several instructions at once rather than only a single one. Of course, Qemu could not be used without modification. First, it had to be transformed from a stand-alone executable into a Windows shared library (DLL), whose exported functions can be used by Anubis. Second, Qemu's translation process was modified such that a callback routine into the analysis framework is invoked before every basic block that is executed on the virtual processor. This allows to tightly monitor the process under analysis. Before a dynamic analysis run is performed, the modified PC emulator boots from a virtual hard disk, which has Windows XP (currently with Service Pack 2) installed. The lengthy Windows boot-process is avoided by starting Qemu from a snapshot, which represents the state of the PC system after the operating system has started.

When Anubis receives new samples, Anubis executes the binary and monitors the invocation of important system and Windows API calls, records the network traffic, and tracks data flows. This provides a comprehensive view of malicious activity that is typically not possible when monitoring network traffic alone. Anubis receives malware samples through a public web interface and a number of feeds from security organizations and anti-malware companies. These samples are collected by honeypots, web crawlers, spam traps, and by security analysts from infected machines. Thus, they represent a comprehensive and diverse mix of malware found in the wild. The system has been live for a period of about four years. During this time, Anubis has analyzed several million unique binaries (based on their

MD5 hashes). Given that processing each malware program is a time consuming task that can take up to several minutes, this amounts to more than hundreds of CPU years worth of analysis. When compiling statistics about the behaviors of malicious code, one has to consider that certain malware families make use of polymorphism. Since samples are identified based on their MD5 hashes, this means that any malware collection typically contains more samples of polymorphicmalware programs than of non-polymorphic families. Unfortunately, this might skew the results so that the behavior (or certain actions) of a single, polymorphic family can completely dominate the statistics. To compensate for this, it analyzes behaviors not only based on individual samples in the database but also based on malware families (clusters).

For each submitted malware sample, an Analysis report is generated and can be accessed publicly. Such a report consists of the following information:

1. General information. This section contains information about TTAnalyze's invocation, the command line arguments, and some general information about the test subject (e.g., size, exit code, time to perform analysis).

2. File activity. This section covers the activity of the test subject (i.e., which files were created, modified,).

3. Registry activity. In this section, all modifications made to the Windows registry and all registry values that have been read by the test subject are described.

4. Service activity. This section documents all interaction between the test subject and the Windows Service Manager. If the test subject starts or stops a Windows service, for example, this information is listed here.

5. Process activity. In this section, information about the creation or termination of processes (and threads) as well as interprocess communication can be found.

6. Network activity. This section provides a link to a log that contains all network traffic sent or received by the test subject.

Once reduced to this human-readable list of modifications caused by an executable on the underlying system, the evaluation is complete and can be used for further processing. Another implementation of this sandboxing-technique is called CWSandbox.

3.2.2 CWSandbox

CWSandbox [173] is a malware analysis tool that fulfills the criteria of automation, effectiveness, and correctness for the Win32 family of operating systems. It uses API hooking and dynamic linked library (DLL) injection techniques to implement the necessary rootkit functionality to avoid detection by the malware. Comparable to Anubis, CWSandbox is capable of tracing the most important system calls and generate a report that describes

* the files the malware sample created or modified;
* the changes the malware sample performed on the Windows registry;
* which DLLs the malware loaded before execution;
* which virtual memory areas it accessed;
* the processes that it created;
* the network connections it opened and the information it sent; and
* other information, such as the malwares access to pro- tected storage areas, installed services, or kernel drivers.

With the *API hooking* and *DLL code injection* as the underlying methods, CWSandbox is capable of automatically analyzing a malware sample. This system outputs a behavior-based analysis; that is, it executes the malware binary in a controlled environment so that all relevant function calls to the Windows API can be observed, and a high-level summarized report from the monitored API calls can be generated. The report provides data for each process and its associated actions, one subsection for all accesses to the file system and another for all network operations, for example. One the focuses is on bot analysis, so a considerable effort is spent on extracting and evaluating the network connection data. After it analyzes the API calls' parameters, the sand- box routes them back to their original API functions. Therefore, it doesn't block the malware from integrating itself into the target operating system by copying itself to the Windows system directory, for example, or adding new registry keys. To enable fast automated analysis, the CWSandbox is executed in a virtual environment so that the system can easily return to a clean state after completing the analysis process.

3.2.3 Related Projects

Other than the previously mentioned examples, several tools exist for automatically analyzing malicious software behaviors. The Norman SandBox[28], for example simulates an entire computer and a connected network by reimplementing the core Windows system and executing the malware binary within the simulated environment. It's also possible to execute the malware binary with a live Internet connection. The companys Web site features implementation details, a description of the underlying technology, and a live demo. Such environments are mostly transparent to the malware, which can't detect that they're being executed within a simulated environment. Yet, simulations dont let the malware processes interfere with, infect, or modify other running processes because no other processes run within the simulation. A different approach is Chas Tomlin's Litterbox [10], in which malware is executed on a real Windows system, rather than a simulated or emulated one. After 60 seconds of execution, the host machine is rebooted and forced to boot from a Linux image. After booting Linux, Litterbox mounts the Windows partition and extracts the Windows registry and complete file list; the Windows partition reverts back to its initial clean state. Litterbox focuses on network activity, so it makes several dispositions of the simulated network. During malware execution, the Windows host connects to a virtual Internet with an IRC server running, which answers positively to all incoming IRC connection requests. The tool captures all packets to examine all other network traffic afterwards.

The Reusable Unknown Malware Analysis Net [30] takes a similar approach. Galen Hunt and Doug Brubacher introduced Detours[99], a library for instrumenting arbitrary Windows functions.

There are, of course, several other related project involving sandboxing techniques to monitor malware behaviour. Most of them, however, function based on the same principles mentioned here. The next section will describe how the gathered information and data about the executables is used for further analysis.

3.3 Post Processing Data

Once, a decent amount of information about a code binary is available, it can be used for further investigation. Running the malware, for instance, does usually not suffice to deduce the malware family a certain sample belongs to. Instead, the available data must first be post-processed, where its behavior can be assessed based on the gained knowledge.

Otherwise, the analyst dealing with the evaluation is facing thousands of reports every day that need to be examined. Thus, there is a need to prioritize these reports and guide an analyst in the selection of those samples that require most attention. One approach to process reports is to cluster them into sets of malware that exhibit similar behavior. The ability to automatically and effectively cluster analyzed malware samples into families with similar characteristics is beneficial for the following reasons: First, every time a new malware sample is found in the wild, an analyst can quickly determine whether it is a new malware instance or a variant of a well-known family. Moreover, given sets of malware samples that belong to different malware families, it becomes significantly easier to derive generalized signatures, implement removal procedures, and create new mitigation strategies that work for a whole class of programs.

Grouping individual malware samples into malware families is an idea proposed by several authors [54, 90, 98, 110, 121]. The main problem of these approaches is, that they generally do not scale well and are slow for the size of malware sets that anti-malware companies are confronted with. To tackle this problem, scalable and precise malware clustering techniques are being developed and implemented [61]. These techniques are based on a dynamic analysis system that monitors the execution of a malware sample in a controlled environment. Unlike systems that operate directly on low-level data such as system call traces, they enrich and generalize the collected data and summarize the behavior of a malware sample in a behavioral profile. These profiles express malware behavior in terms of operating system (OS) objects and OS operations. Moreover, profiles capture a more detailed view of network activity and the ways in which a malware program uses input from the environment. This allows these systems to recognize similar behaviors among samples whose low-level traces appear very different. Finally, the analyzed samples are clustered according to their behavioral profile.

After this post-processing step, the information is ready to be shared with interesting parties. At this point, the arguments mentioned in 3 are gaining momentum again. In the end, the sharing policy can decide if an organization is rated as black- or whitehat.

3.4 Sharing Data

With an available dataset about a multitude of malicious code sample, one would think that sharing this information among interested parties is of the utmost importance for everyone involved. Unfortunately, the solution is not as simple as providing a publicly available service which is free for everyone to access. The main difficulties when sharing data from an automatic malware analysis system are the following:

- Who do i share my information with? Allowing an unrestricted access to malware analysis results will sooner or later attract malware writers. As they are constantly pressured to develop new methods to avoid AV detection, a public environment to test their newest evasion technique is more than welcome.

- Who is allowed to submit new samples? Apart from the blackhat-whitehat problem, dynamic analysis of unknown binaries is always a costly endeavor. Ressources are limited and have to be split to make sure that 0-day malware is processed in a timely manner while still making sure that submissions are not rejected because of a ressource bottleneck

- A common interface has to be found to allow all interested parties a (possibly secured) interaction with the underlying database, or to accumulate multiple reports into a single one.

Several different approaches exists, some of them varying even in their fundamental approach. Some of them are listed in the following Sections.

3.4.1 Virustotal

Virus Total [33] is a perfect example of an approach to bring together knowledge produced by various sources. It is a service developed by Hispasec Sistemas that analyzes suspicious files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and web analysis toolbars.

In its core, Virus Total is a large-scale malware scanner which runs a multitude of AV solutions to test a binary or a URL for malicious activities. Here, the arguments mentioned in the previous section apply. A counter-argument is, however, that the built-in API functionality also enables AV companies and security researchers to automatically harvest data and use it to improve security for their end-users. Still, potential miscreants can use it to check if they are successfully avoiding most common AV scanners and malware detection techniques.

3.4.2 Predict

PREDICT [20], the **P**rotected **Re**pository for the **D**efense of **I**nfrastructure Against **C**yber **T**hreats, is a community of producers of security-relevant network operations data and researchers in networking and information security. Through a centralized repository, it provides developers and evaluators with regularly updated network operations data relevant to cyber defense technology development. Initiated by the US Department of Homeland Security (DHS), Predict was established as a repository for current computer and network operational data.

Network-related Datasets are made available to qualified cyber defense researchers to help them create and develop new models, technologies, and products to assess cyber threats to the country's computing infrastructure and increase their security capabilities. To ensure that business intelligence and individual privacy are not compromised by sharing these datasets, Predict has established a data sharing protocol. Key elements of the protocol are:

- Access requirements are established through data sensitivity assessments.

- Access permission is granted after review and approval by independent experts and data provider(s).

- Data usage is subject to legally binding terms and conditions.

Researchers use the Predict data catalog to determine the most applicable dataset for their research and apply for access to the dataset(s). Information on the type of data, time frame for the data snapshot, requirements and restrictions for using the data, as well as criteria for transmission, are provided for each dataset. Multiple datasets may be requested as part of a dataset application. In the case of multiple requests, all requirements for all datasets must be met. Applications will be reviewed by the PREDICT Application Review Board.

Data Providers use the PREDICT portal to upload and register new datasets or retire existing datasets. They specify conditions under which researchers can use their datasets and provide the documents necessary for Researchers to gain approval to use their data. Data Providers also use the portal to monitor who has requested their datasets and communicate their decisions on whether to allow them to be used.

To finally gain access to data, users must be registered users and must submit an application for access to specific datasets. They must also agree to the conditions of use specified for the datasets they request, and they must specify the research they will be conducting using the requested datasets. All applications for datasets are reviewed by the Application Review Board. This way, the members try to ensure that only approved individuals gain access to the sensitive information provided by these datasets.

3.4.3 WOMBAT - WAPI

A similar approach is implemented by WAPI, the API is being developed in line with the EU Project Wombat(**W**orldwide **O**bservatory of **M**alicious **B**ehaviors and **A**ttack **T**hreats) [35].

The general idea here is, to tackle privacy and confidentiality issues which limit the sharing of information between sources. These limitations have prevented the emergence of an open standard investigation framework for consistent and systematic malware analysis. Rather than just sharing data among participants, the project aims at providing new means to understand the existing and emerging threats to the Internet infrastructure and the services this infrastructure supports.

The WOMBAT project is organized around the following activities:

- Real-time gathering of a diverse set of security-related raw data. WOMBAT takes advantage of existing data collection efforts undertaken by its partners and collaborating organizations.

- Data enrichement by means of various analysis techniques. WOMBAT formalizes threat context information.

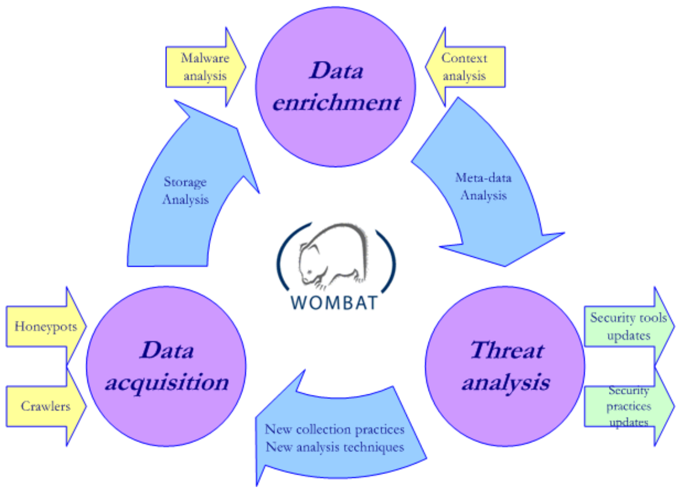


Figure 3.1: The WOMBAT project

- Threat analysis. WOMBAT builds upon the information correlation expertise of its partners to provide root cause analysis.

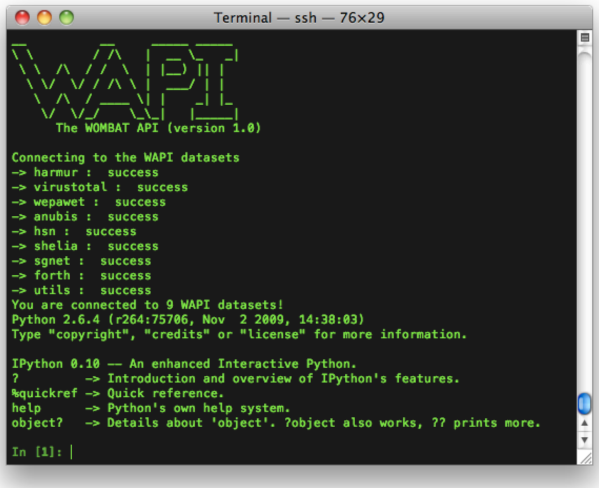
The key features of the project are depicted in Figure 3.1

The acquired knowledge is shared with all interested security actors (ISPs, CERTs, security vendors, etc.), enabling them to make sound security investment decisions and to focus on the most dangerous activities first. Special care will also be devoted to impact the level of confidence of the European citizens in the net economy by leveraging security awareness in Europe thanks to the gained expertise.

To enable all participants a convenient integration of both, their available datasets and the tools to use them, the Wombat API (WAPI) was devised.

The WAPI provides a set of primitives to offer to clients an object-oriented view of any given dataset. All the primitives are based on the concept of objects. An object is identified univocally by a type (e.g. ”malware”, ”event”, ”source”) and by an identifier. The identifier is an opaque identifier that is used to identify the object in your dataset. For instance, a malware sample might be identified by its MD5 hash, but it could also be identified by an opaque ID that corresponds to the primary key used inside your dataset. This flexibility ensures, that all datasets, independent from their content, can be mapped to a globally valid data structure.

WAPI features an implementation of available datasets via various methods. Once, all participants linked their datasets, they can be accessed and used for further processing. Figure 3.2 shows a screenshot of the WAPI when its connected to all available data sources.



```
__          __     _____ _____
\ \        / /\   |  __ \_   _|
 \ \  /\  / /  \  | |__) || |
  \ \/  \/ / /\ \ |  ___/ | |
   \  /\  / ____ \| |    _| |_
    \/  \/_/    \_\_|   |_____|
      The WOMBAT API (version 1.0)

Connecting to the WAPI datasets
-> harmur :  success
-> virustotal :  success
-> wepawet :  success
-> anubis :  success
-> hsn :  success
-> shelia :  success
-> sgnet :  success
-> forth :  success
-> utils :  success
You are connected to 9 WAPI datasets!
Python 2.6.4 (r264:75706, Nov  2 2009, 14:38:03)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: 
```

Figure 3.2: The WOMBAT API (WAPI)

3.5 Conclusion

Today, malware analysis is in a very advanced state. While certainly not perfect in certain respects, it provides valuable information for security researchers and AV companies alike. Whats missing is a reliable, secure and feasible way to relay this information to interested parties, while keeping it from people with malicious intend at the same time. The initiatives presented in this chapter aim to solve this problem on different levels, starting from a free-for-all approach to the point of serious identity checks.

In the end, the fundamental consensus is, that it is an imperative requirement to share malware-related information an accumulate gained knowledge to create comprehensive datasets for both, security researchers and AV companies alike.

4 Online Fraud

The term *online fraud* or *internet fraud* comprises all sorts of illicit practices carried out to conduct fraudulent transactions (e.g., earning money illicitly) through the abuse of Internet services such as web services, websites, chat rooms, forums, or online auction sites with. These practices are manifested in several different ways including, among others, the following famous schemes: stock market manipulation schemes, automotive selling scams, counterfeit postal money orders fraud, shipping scams, "work-at-home" offers, dating fraud, and, to some extent, credit card theft and card skimming. All these practices participate to some extent to online fraudulent activities.

Online fraud itself is "enabled" by other malicious practices such as phishing, spamming, malware and rogue software campaigns. Fortunately, online fraud is somewhat mitigated by countermeasures such as phishing detection and spam filtering, and can be effectively combated by tangential approaches that aim to break the vicious circle (see for example Figure 4.1) at its source (e.g., malware detection to contain infections, web application protection to prevent the miscreants from compromising a website to install malicious scripts and other drive-by-download threats, such that visitors will fall victims).

In this section we describe the online fraud landscape. First, in Section 4.2 we explain the new forms of phishing, along with social engineering, and how they contribute to online fraud practices. Then, in Section 4.3 we describe upcoming threats such as click fraud, advertisement fraud, and other "Web 2.0"-style fraud schemes for which only mitigation countermeasures exist.

4.1 Introduction

The explosion of online social networks confirms that people tend to move their social interactions (e.g., status updates, instant messaging) into cy-

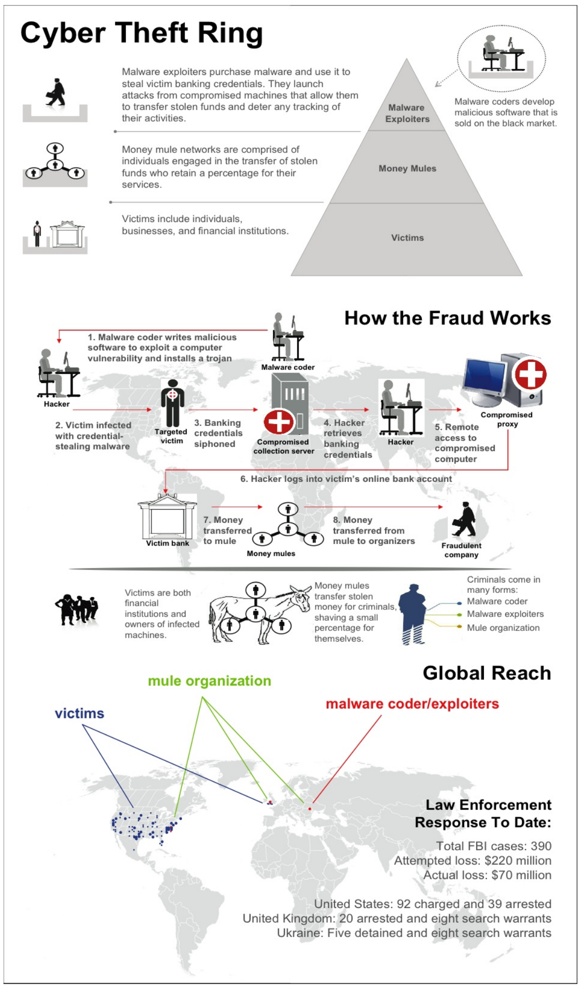

Cyber Theft Ring
Malware exploiters purchase malware and use it to steal victim banking credentials. They launch attacks from compromised machines that allow them to transfer stolen funds and deter any tracking of their activities.
Money mule networks are comprised of individuals engaged in the transfer of stolen funds who retain a percentage for their services.
Victims include individuals, businesses, and financial institutions.
Malware Exploiters
Money Mules
Victims
Malware coders develop malicious software that is sold on the black market.
How the Fraud Works
1. Malware coder writes malicious software to exploit a computer vulnerability and installs a trojan
Hacker
Malware coder
2. Victim infected with credential-stealing malware
Targeted victim
3. Banking credentials siphoned
Compromised collection server
4. Hacker retrieves banking credentials
Hacker
Compromised proxy
5. Remote access to compromised computer
6. Hacker logs into victim's online bank account
Victim bank
7. Money transferred to mule
Money mules
8. Money transferred from mule to organizers
Fraudulent company
Victims are both financial institutions and owners of infected machines.
Money mules transfer stolen money for criminals, shaving a small percentage for themselves.
Criminals come in many forms:
Malware coder
Malware exploiters
Mule organization
Global Reach
mule organization
victims
malware coder/exploiters
Law Enforcement Response To Date:
Total FBI cases: 390
Attempted loss: $220 million
Actual loss: $70 million
United States: 92 charged and 39 arrested
United Kingdom: 20 arrested and eight search warrants
Ukraine: Five detained and eight search warrants

Figure 4.1: The big picture of a modern fraud scheme explained through the case study of Zeus (source: FBI).

berspace, and to increasingly make use of new communication opportunities. For example, services such as voice over IP (e.g., Skype) are extremely popular. Unfortunately, online trust relationships are fertile lands for fraud. We can observe the aforementioned shift in phishing and related scams. Phishers nowadays rely on a variety of channels, ranging from old-fashioned emails to instant messages, social networks, and the phone system (with both calls and text messages), with the goal of reaching more victims. As a consequence, modern phishing is a multi-faceted, even more pervasive threat that is inherently more difficult to study than traditional, email-based phishing.

Cyber criminals have gathered together and established a flourishing underground economy, which operates on public communication servers (e.g., IRC, forums) and actively violate the laws of several nations and the rights of individuals [38]. Modern cyber criminals are well-organized and profit-driven, as opposed to the reputation-driven underground which was prevalent years ago [75, 129, 69]. Evidence about such shift is available today to researchers. For example Fallmann et al. in [83] describe a system which automatically monitors underground marketplaces (e.g., forums, IRC channels), to collect precious information about how the cyber criminals exchange information and goods. Another study along this line, which has helped substantially at understanding the economy of spam as a fraudulent activity has been presented by Stone-Gross et al. in [161], where they analyze a spam campaign from the perspective of a botnet that they have managed to take over; they estimate that the revenue of the spam campaign made with the Cutwail botnet is between between 1.7 and 4.2 million dollars over one year.

4.2 Phishing and Spam Enable Frauds

Phishing is perhaps the major "technology" employed by cyber criminals to conduct fraud (e.g., stealing money, reselling credentials on the black market). Phishing is the practice of eliciting a person's confidential information such as name, date of birth or credit card details. Typically, the phishers combine some technologies and simple social engineering stratagems to persuade the victims into voluntarily disclose sensitive data. Phishing based on spam email is certainly the most popular form, for which no definitive countermeasures exist. However, researchers have been recently leveraging vantage observation points (e.g., by infiltrating into botnets [160, 71]) for tracking the origins of and the economical driving spam and phishing campaigns.

In [125] the authors use various data sets to analyze aspects of the modus operandi of phishers. They examine the anatomy of phishing URLs and domains, they analyze the lifecycle of domains and hosting machines, and propose heuristics to filter phishing-related emails and to identify suspicious domain registrations.

In [115, 116] the authors infiltrated the Storm botnet and analyzed ten months of spam campaigns conducted through it. They analyzed the raw templates of spam campaigns (identifying over 90 different campaign types, targeting over 630 million different email addresses and using more than 90,000 spamming zombies). They classified the topics of campaigns, observed their temporal behavior, and studied evasion techniques. But more importantly, they made significant, data driven observations on the *marketing strategies* employed by spammers. A similar approach has also been proposed as a way to improve spam filtering [141]

4.2.1 Beyond Email Spamming and Phishing

The miscreants are obviously aware of the increased knowledge that researchers have accumulated in recent years to fight email spam and other related practices. During the past five years, however, the blackhats have begun to leverage alternative spreading channels such as instant messaging [112, 82], forums, blogs and even short text messages on mobile phones [97, 153].

4.2.1.1 Online Social Networks

In [154] the authors investigate and compare automated tools used to spam forums, in particular XRumer, a widely used example. They find that the software has been designed to get around many practices used by forums to distinguish humans from bots, while keeping the spammer anonymous.

Grier et al. analyzed spam messages on Twitter. Although their work [92] is restricted to Twitter, they were among the first who analyzed spamming activities on social networks. Stringhini et al. developed a system, described along with their findings in [163], to automatically detect spammers on Twitter and Facebook. Gao et al. in [89] have instead focused on Facebook wall posts: they analyzed the activity of about 3.5 million users and detected approximately 200,000 rogue posts (out of 187 million posts) with embedded URLs driving users to malicious or spam content.

4.2.1.2 Instant Messaging

Back in 2005 Yahoo!'s free instant messaging service was confirmed by Yahoo! of being targeted by phishers to steal usernames, passwords and other personal information, via social engineering-based scams [112]. Attackers were spreading messages containing a link to a fake web site that looked like an official Yahoo! site and asks the user to log in by entering their Yahoo! ID and password. A similar attack has been reported one year later [82]. Antonatos et al. in [48] have recently deployed a multi-protocol instant message honeypot that they used to conduct a thorough research that assessed the extents of this threat. From over six thousand contacts, they received messages worth 50 to 110 malicious URLs per day: 93 percent of the identified IM phishing domains were not recorded by popular blacklist mechanisms.

4.2.2 Automated Social Engineering Techniques

As a part of their modern arsenal, the miscreants have learned to streamline their campaigns also by leveraging automated social engineering attacks over several communication channels, with the goal of expanding their "business" beyond email users. These "new" flavors of online fraud resemble traditional *a-lá-Mitnick* scams, which are based on pure social engineering techniques and, despite their effectiveness, they are relatively slow. To make this a viable business, modern scammers have begun to take advantage of the customers' familiarity with "new technologies" such as Internet-based telephony, text-messages, and automated telephone services. Another example is the use of instant messaging (e.g., Windows Live Messenger, Skype, FaceBook chat), which involves some form of conversation with computer programs that leverages natural language processing and artificial intelligence techniques to mimic a real person. For example, in [120] the authors raise the concern that automated social engineering is a serious information security threat to human communications on the Internet, because the attacks can easily scale to a large number of victims. They were indeed able to implement with success a new attack that instruments human conversations (e.g., online chats) for social engineering, or spamming. They managed to fool current detection mechanisms, and achieved a link click rate up to 76.1%. Like many other classes of attacks, large-scale social engineering attacks are also facilitated by the availability of software toolkits that can automate many of the steps required for such an attack, such as the freely available Social Engineer Toolkit [26]. For instance, this toolkit includes a "site cloning" module that is able to automatically create a replica of a target website for the purpose of phishing, and an "infectious media" module that can be used to prepare removable media that will infect a computer it is inserted into by taking advantage of autorun functionality.

4.2.2.1 VoIP and Other Voice Media

The criminals are also resorting to the telephone channel to achieve the same objectives of traditional e-mail phishing. This practice is referred to as *vishing* (VoIP phishing, or, in its most generic meaning, voice phishing) [135]. More precisely, vishing is the activity of systematically defrauding account holders using social engineering over the telephone system. Real-world episodes from the past have extensively proven how successful live telephone calls can be [127]. A live conversation gives the victim less chances to think through what is happening and what he is required to do. This does not happen normally with email-based phishing because emails have to be read through. Similarly to phishing, the goal of vishing is to eliciting a person's confidential information, although it emerged from [123] that most of the times these "vishers" attempt to elicit credit card information, with the goal of stealing money directly from the victims. In other similar, voice-based scams, the victims are lured with some excuses to wire money directly to the miscreants. Unfortunately, vishing is inherently more difficult to analyze as opposed to traditional, e-mail phishing. In fact, collecting e-mails suspected of phishing is relatively easy. For instance, from a purely technical perspective, e-mails can be intercepted, dispatched, filtered, stored and so forth. Vishing was popular in the U.S. in 2006–2009 [100], and is now slowly gaining ground in Europe. Notably, an experiment conducted in 2010 by the United Nations Interregional Crime and Justice Research Institute revealed that the 25.9% of Italians (on a sample comprising 800 randomly-selected citizens) were successfully tricked by phone scammers. In [123] the authors analyzed this type of scams, based on a selection of about 400 user-submitted reports, including the caller identifier (e.g., source phone number), (parts of) the transcribed conversation, general subject of the conversation, and spoken language. Besides confirming that vishing was popular in the U.S. at that time, the preliminary results suggests that phishers rely on automated responders, and not only on live calls, with the goal of reaching a broader spectrum of victims. Reports were filed between 2009 and 2010 through `http://phonephishing.info`, a publicly-available website where anyone can submit anonymous reports of vishing: given the difficulty of gathering evidence for studying this phenomenon, this website allows users to report suspicious phone calls, completely anonymously. Along this line, the authors have recently proposed in [124] a data collection system to capture different aspects of phishing campaigns, with a particular focus on the emerging use of the voice channel. In fact, another more recent type of scam that involves voice communication is what can be referred to as "reverse vishing": Instead of calling the victims directly, the cyber criminals run phishing campaigns with emails that claim some interesting business offers for which telephone contact is provided. The general approach proposed in [124] to study this phenomenon is to record inbound calls received on decoy phone lines, place outbound calls to the same caller identifiers (when available) and also to telephone numbers obtained from different sources. Specifically, their system analyzes instant messages (e.g., automated social engineering attempts) and suspicious emails (e.g., spam, phishing), and extracts telephone numbers, URLs and popular words from the content. In addition, users can voluntarily submit voice phishing (vishing) attempts through the existing public website. Extracted telephone numbers, URLs and popular words are then correlated to attempt to recognize campaigns by means of cross-channel relationships between messages.

In addition to the abuse of the voice channel for luring victims, the telephone system, and in particular VoIP infrastructures, can be exploited to keep telephone devices busy, hindering legitimate callers from gaining access. This kind of attack—demonstrated in its most stealthy variant by Kapravelos et al. in [105]—is very concerning, as several online banking transactions are carried out over the phone, or involve outbound verification steps that require the initiator to call or receive a call from a number for completing, for example, a wire transfer.

The countermeasures against this emerging threat are limited to a small number of approaches. For example, in [175] the authors proposed a system to mitigate the threat posed by malicious SMS [153, 97] received by customers via anomaly detection techniques. A recent research shows that it is possible to determine whether a caller (e.g., a scammer) is using VoIP or not [56]; this result may stimulate further research toward devising effective detection systems against attacks that target VoIP and other voice media. However, we notice a complete lack of definitive countermeasures.

4.3 Fraud 2.0

In this section we describe upcoming threats such as click fraud, advertisement fraud, and other "Web 2.0"-style fraud schemes. We also survey the existing mitigation approaches, which are unfortunately far from being definitive countermeasures.

4.3.1 From Hit Inflation to Advertisement Fraud

The idea of artificially inflating the "hits" of a web object has been at least as old as the association of value to hits. In fact, the seminal paper on hit inflation [49] describes how an attack allows the referrer to transform every visit by a user on any site that is collaborating with the referrer into a click through to the target.

Of course, this became more and more relevant as "clicks" transformed into "money" in the modern Internet economy. After about ten years, researchers came up with detection schemes to detect hit inflation [126, 178].

Click fraud, however, still exists and is a major threat. The miscreants for instance turned clickjacking into a tool for generating revenue. Clickjacking refers to a set of techniques used to "steal" clicks from the users without their consent. In other words, with clickjacking the attacker "tricks the user" into unwillingly clicking on one or more links. In [27, 5] clickjacking is demonstrated on Twitter, but many other websites are also vulnerable to this threat [152].

The fast way to monetize clickjacking is tricking users into initiating money transfers, clicking on banner ads to generate revenue ("advertisement click fraud" [88, 130]), or, in general, into performing any action that can be triggered by a mouse click [57].

Economic models have been developed to estimate the impact of click frauds in general [118], as well as the impact of advertisement frauds specifically [118, 130].

Prevention is possible through anomaly detection [57], but unfortunately a recent study demonstrated that modern clickbots (i.e., malware specialized for conducting click fraud) adopt quite sophisticated evasion techniques [84].

4.3.2 Rogue Software and Scareware

Scareware is a class of software that usually has no malicious payload and thus fly under the malware detectors' radar. The purpose of scareware is to earn money directly from the users, by convincing them via social engineering that their computer has been somehow infected or compromised, so that to ease the task of selling them a (fake) cleaning software. In some cases, scareware is combined with malware infection to actually bring the system in unstable states, such that victims are forced to purchase the advertised software to unlock their computers. This type of software is also known as "rogue antivirus" or "fake antivirus". The miscreants also came up with "affiliate programs" that pay people to resell worthless or rogue software [113]. In 2009, reports have estimated that one of the most active affiliates of TrafficConverter.biz (one of such affiliate programs) has earned up 330,000 dollars per month in commissions. The spread of scareware is partly being driven by search engine poisoning techniques, used to direct surfers to sites peddling scareware.

The problem of rogue software was identified first in a Microsoft Research paper [134] by O'Dea H., who examined the shifts in the rogue landscape and compared their evolution to that of other types of malware. Rajab et al. in [148] present their study on rogue antivirus from the perspective of Google. Over a 13 month period discovered over 11,000 domains involved in fake antivirus distribution, out of 240 million web pages collected by Google's malware detection infrastructure. The authors show that the fake antivirus threat is rising both absolutely and relative to other forms of web-based malware. Cova et al. in [72] report the results of a broad study on this threat. They were able to track 372,096 victims over a period of 2 months: In the worst-case scenario, the authors estimate that the gross income is about 21,000 to 35,000 dollars. Recent studies have also analyzed and modeled the economic system that lies behind the rogue-antivirus phenomenon [159].

4.4 Other and Upcoming Frauds

Emerging technologies such as electronic-payment services, electronic and mobile commerce, cyber-payment, mobile banking and pay-as-you-go insurance services are opening up new avenues for criminals to commit computer-related financial fraud and online abuse. For example, card skimming, ATM and physical credit card fraud, despite not strictly related to "internet fraud" (and thus, for instance, not displayed in the vicious cycle depicted in Figure 4.1), have a significant role in frauds.

Multiple attacks have also been developed against contactless payment systems [107], which are of growing importance and deployment (not just in mobility cards, but also for real payment applications).

Another direction is the one shown in [179], which demonstrates the practical feasibility of attacking the billing scheme employed by SIP-based VoIP systems, resulting in charges on calls the subscribers are not responsible for, or in overcharges on VoIP calls the subscribers have made. Also, Avoine G. in [50] describes a protocol by which two parties can place free telephone calls without being noticed. In light of these attacks—combined with the widespread of VoIP telephony, we believe that the cybercriminals can effectively employ such tool to earn money illicitly.

4.5 Lack of Research Data Collection Initiatives

From all the researches that we have reviewed in this chapter it emerges that, although it is possible to collect evidence about online fraud, automating the collection of this type of data is inherently a difficult task. In fact, we have seen that authors have resorted to very different methods to collect data for the purpose of each research. This suggest the need for collaboration and data-sharing efforts toward the creation of centralized repositories of structured and unstructured evidence about these threats.

There are, however, a number of non-research initiatives that strive to collect evidence about online fraud directly from the end-users (i.e., the victims), although collecting data directly from the crowds may suffer from skews caused by the most active submitters, as noticed in [128] in the case of phishing. In particular, they have studied the PhishTank database—the most famous initiative to collect user-submitted phishing emails—and showed that the activity of the users follow a power low distribution. There are also other examples of such data collection and sharing initiatives, along the line of IPInfoDB, which offer a free and publicly available fraud detection verification procedure to check whether an online ecommerce transaction, for instance, involves a credit card suspected of fraudulent activity; it is not clear, however, what are the feeds of their database. Another notable example is the spamdetector service, developed by the authors of [163] along with a Twitter API available at `http://twitter.com/#!/spamdetector`: Users who want to report or check whether a certain Twitter account is sending out spam can contact the automated bot via Twitter messages.

5 Network

Previous Chapters of this Deliverable already focused on network-related data collection initiatives. For example, Honeypots are one of the most well known and largely adopted technique to collect data related to attacks against a network. Similar considerations can be made for Online Fraud that, as the name said, is related to “online” phenomena.

This Chapter focus on three main topics related to network collection in malware and fraud that were not discussed in the previous sections. In particular, it presents an overview of past research in the areas of *botnet* and *worm* detection, and on DNS analysis for detecting domain used in malicious activities. The focus of this survey is on the collected datasets that were used by the authors to test and evaluate the proposed techniques.

5.1 Botnet Detection

Bots are a popular tool of choice for criminals. A bot is a type of malware that is written with the intent of compromising and taking control of hosts on the Internet. It is typically installed on the victim’s computer by either exploiting a software vulnerability in the web browser or the operating system, or by using social engineering techniques to trick the victim into installing the bot herself. Compared to other types of malware, the distinguishing characteristic of a bot is its ability to establish a command and control (C&C) channel that allows an attacker to remotely control or update a compromised machine. A number of bot-infected machines that are combined under the control of a single, malicious entity (called the botmaster) are referred to as a *botnet*. Such botnets are often abused as platforms to launch denial of service attacks, to send spam mails, to collect private user data, or to host scam pages.

Horizontal Correlation

A number of botnet detection systems perform horizontal correlation on network data. That is, these systems attempt to find similarities between the network-level behavior of hosts. The assumption is that similar traffic patterns indicate that the corresponding hosts are members of the same botnet, receiving the same commands and reacting in lockstep. While initial detection proposals [95, 106] relied on some protocol-specific knowledge about the command and control channel, subsequent techniques [93, 149] removed this shortcoming.

BotSniffer [95] looks for infected hosts displaying spatial-temporal similarity in their network activities. For the paper's experiments, the authors used multiple network traces captured from their university campus network (GaTech). Among those, eight were just port 6667 IRC traffic captured in 2005, 2006, and 2007. Each IRC trace lasted from several days to several months. The total duration of these traces was about 189 days. The other five traces were complete packet captures of all network traffic. Two of them were collected in 2004, each lasting about ten minutes. The other three were captured in May and December 2007, each lasting 1 to 5 hours. The primary purpose of using these traces was to test the false positive rate of BotSniffer.

The authors also obtained several real-world IRC-based botnet C&C traces from several different sources. One was captured using an honeypot in June 2006. This trace contained about eight hours of traffic (mainly IRC). The IRC channel had broadcast on and they could observe the messages sent from other bots in the channel. Additionally, they also obtained two botnet IRC logs (not network traces) recorded by an IRC tracker in 2006. In these logs, there were two distinct IRC servers, so there were two different botnets.

In addition to these IRC botnet traces, they modified the source codes of three common bots (Rbot, Spybot, Sdbot) and created their version of binaries (so that the bots would only connect to their controlled IRC server). The authors set up a virtual network environment using VMware and launched the modified bots in several Windows XP/2K virtual machines. They instructed the bots to connect our controlled C&C server and captured the traces in the virtual network. For Rbot, they used five Windows XP virtual machines to generate the trace. For Spybot and Sdbot, they used four clients.

Karasaridis et. al. [106] use primarily transport layer flow summary data for identification of botnet controllers. In comparison to packet-level analysis, flow data reduces some privacy protection concerns. Flow data also significant reduces the amount of data to process, which makes it practical to transport data to a central location for cross-correlation. The implementation is scalable to large networks in comparison to packet-level analysis since nearly all network devices can generate at least sampled flow data without significant performance impact or modification. In their application, the authors have invested in the capability to generate unsampled data in select portions of the network to complement more sparsely sampled flow data across a large Tier 1 ISP network. The implementation collects many billions of flow records each day for security analysis in addition to botnet identification processing. Flow records contain summary information about sessions between a single source address/port (sip/sport) and a destination address/port (dip/dport) using a given protocol. A single flow record contains the number of packets, bytes, and an OR function of the flags used (if TCP is the transport layer protocol), the start and end time of the session and the transport layer protocol used. Flow record data are collected from a large number of geographically and end-point diverse circuits to a central processing facility where they can be filtered, processed, and correlated.

BotMiner [93] tries to group host based on their destination and connection statistics and on a number of suspicious activities (e.g., network scans, download of binaries, and SPAM). The authors set up traffic monitors to work on a span port mirroring a backbone router at the campus network of the College of Computing at Georgia Tech. The traffic rate was typically around 200Mbps-300Mbps at daytime. They ran the monitors for a 10-day period in late 2007. A random sampling of the network trace showed that the traffic was very diverse, containing many normal application protocols, such as HTTP, SMTP, POP, FTP, SSH, NetBios, DNS, SNMP, IM (e.g., ICQ, AIM), P2P (e.g., Gnutella, Edonkey, bittorrent), and IRC. This served as a good background to test the false positives and detection performance on a normal network with rich application protocols.

For the paper, the authors also collected a total of eight different botnets covering IRC, HTTP, and P2P traffic. They re-used two IRC and two HTTP botnet traces introduced in previous research [95], i.e., V-Spybot, V-Sdbot, B-HTTP-I, and B-HTTP-II. In short, V-Spybot and V-Sdbot were generated by executing modified bot code (Spybot and Sdbot) in a fully controlled virtual network. The experiments included four Windows XP/2K IRC bot clients, and last several minutes. The clients communicated with a controlled server and executed the received commands (e.g., spam). In B-HTTP-I, the bot contacted the server periodically (about every five minutes) and the whole trace lasted for about 3.6 hours. B-HTTP-II had a more stealthy C&C communication where the bot waits a random time between zero to ten minutes before it visited the server, and the whole trace lasted for 19 hours. In addition, the paper also used a new IRC botnet trace that lasted for a longer time (a whole day), generated using modified Rbot source code. This also was generated in a controlled virtual network with four Windows clients and one IRC server.

Finally, they also obtained two real-world network traces. The first was an IRC-based botnet C&C trace that was captured in the wild in 2004. The trace contained about 7-minute IRC C&C communications, and has hundreds of bots connected to the IRC C&C server. The second was a trace containing two P2P botnets, Nugache and Storm. The trace lasts for a whole day, and there are 82 Nugache bots and 13 Storm bots in the trace. It was captured from a group of honeypots running in the wild in late 2007.

Vertical Correlation

The main limitation of systems that perform horizontal correlation is that they usually need to observe multiple bots of the same botnet to spot behavioral similarities.

A second line of research explored instead vertical correlation, a concept that describes techniques to detect individual bot-infected machines. One system, called Rishi [91], attempts to detect bots based on the structure of nicknames in IRC traffic. Other techniques [68, 162] aim to identify suspicious IRC connections based on traffic properties. In all cases, the detection approaches focus specifically on botnets that use IRC for their command and control. The most advanced system is BotHunter [94], which correlates the output of three IDS sensors – Snort [150], a payload anomaly detector, and a scan detection engine. For their experiments, the authors adopted a VMware-based honeynet deployed at SRI for a three-weeks period between March and April 2007. They also tested BotHunter for five months in their university network on a link that exhibited typical diurnal behavior and a sustained peak traffic of over 100 Mbps during the day.

Wurzinger et. al. [174] performs vertical correlation as well. They present a system that aims to detect bots, independent of any prior information about the command and control channels or propagation vectors, and without requiring multiple infections for correlation. The system relies on detection models that target the characteristic behavior of every bot, the fact that it receives commands from the botmaster to which it responds in a specific way. A key feature is that these detection models are generated automatically. To this end, the system observes the network traffic that is generated by actual bot instances in a controlled environment. The 416 different bot samples were obtained through Anubis [60]. The collection period was more than 8 months. All bot samples were executed in the environment, each producing a traffic trace with a length of five days. In these traffic traces, they first identify points in time that likely correspond to response activity. Then, they extract the corresponding commands that trigger these activities.

When recording bot traffic traces, it was difficult to predict the required amount of time to obtain a representative collection of bot commands and according responses, since it depends on the degree and kind of activity of the botmaster during the observation period. In their experiments, they decided to aim for a capture period of five days. This ensures that they have a good chance of observing a large variety of different commands. However, most bots receive (common) commands after a short time, often within minutes. Thus, they can start to produce a first set of detection models quickly. Then, they wait for several days to capture less frequent commands as well.

Because of the long runtime of the bots as well as our desire to collect as many traces as possible, an important goal when designing the execution environment was to support as many parallel bot instances as possible. They set up a VMware environment on a server with two Intel Xeon 1.86GHz Quad-core processors, 8 GB of memory, and 300 GB of Raid5 disk space. Each VM is running a fully-patched instance of Windows XP with service pack 2, and is able to run with as little as 64 MB main memory. Using this setup, they are able to simultaneously run up to 50 virtual machine instances on their server.

Each of the guest virtual machines is assigned a static, public IP address, and infected with one bot. All network traffic is captured on the host. Since there are no other applications that run and generate network traffic, the bot accounts for all observed network traffic under its host VM's IP address. Of course, a bot requires Internet connectivity, so that it can connect to the command and control infrastructure and receive commands from the botmaster. However, at the same time, they do not wish the bots that they are analyzing to engage in serious and destructive malicious activity such as denial of service attacks. Thus, they had a firewall that rate-limits all outbound network traffic. After each five days capturing period, all VMs are deleted and recreated in a clean state, before the next set of bot samples is executed.

The fact that they use VMware to execute bots could be considered a potential limitation. It is well-known that VMware is easy to fingerprint, and they are aware that a bot could detect their system. However, the problem of VMware detection is not a conceptual limitation of their approach.

5.2 Worm Detection

Network worms are malicious programs that spread automatically across networks by exploiting vulnerabilities that affect a large number of hosts. Because of the speed at which worms spread to large computer populations, countermeasures based on human reaction time are not feasible.

A number of approaches have been proposed that aim to detect worms based on network traffic anomalies. One key observation was that scanning worms, which attempt to locate potential victims by sending probing packets to random targets, exhibit a behavior that is quite different from most legitimate applications. Most prominently, this behavior manifests itself as a large number of (often failed) connection attempts [167, 170]. The first work did not collect any realworld additional data, while the second one was tested on hour-long traces of packet header collected at the access link at the Lawrence Berkeley National Laboratory. This gigabit/sec link connects the Laboratory's 6,000 hosts to the Internet. The link sustains an average of about 50-100 Mbps and 8-15K packets/sec over the course of a day, which includes roughly 20M externally-initiated connection attempts (most reflecting ambient scanning from worms and other automated malware) and roughly 2M internally-initiated connections.

Other detection techniques based on traffic anomalies check for a large number of connections without previous DNS requests [172], or a large number of received ICMP unreachable messages [65].

In addition, there are techniques to identify worms by monitoring traffic sent to dark spaces, which are unused IP address ranges [52], or honeypots [78]. Distributed approaches, like the ones of Bailey et al [52] and Rajab et al. [147] rely on the aggregated information extracted from a distributed set of sensors. In the Rajab's paper, the authors tested their approach using traffic traces from over 1.5 billion suspicious connection attempts observed by more than 1600 intrusion detection systems dispersed across the Internet.

Following a different approach, Kruegel et al. [117] present a technique based on the structural analysis of binary code. Their solution allows one to identify structural similarities between different worm mutations. The approach is based on the analysis of a worm's control flow graph and introduces an original graph coloring technique that supports a more precise characterization of the worms structure. The proposed technique was evaluated on a dataset consisting of 35.7 Gigabyte of network traffic collected over 9 days on the local network of the Distributed Systems Group at the Technical University of Vienna. This evaluation set contained 661,528 total network streams and was verified to be free of known attacks.

Finally, also traditional anomaly detection techniques have been applied to the detection of worms. For example, Agosta et al, [45] present an adaptive end-host anomaly detector based on a supervised classifier that uses a dynamic threshold value that adjusts to track the traffic.

Automatic Signatures Generation

A number of techniques have been proposed to automatically extract worms signatures from network traffic. The first system to automatically extract signatures from network traffic was Honeycomb [114], which looks for common substrings in traffic sent to a honeypot. The system was tested on an unfiltered cable modem connection in three consecutive sessions, covering a total period of three days.

Earlybird [156] and Autograph [108] extend Honeycomb and remove the assumption that all analyzed traffic is malicious. Instead, these systems can identify recurring byte strings in general network flows. Earlybird was evaluated on the traffic flowing in and out of a Local Area Network (LAN) comprised of a total of 7 hosts. Autograph was tested on three packet traces from the DMZs of two research labs; one from Intel Research Pittsburgh (Pittsburgh, USA) and two from ICSI (Berkeley, USA), respectively a T1 and 100 Mbps fiber connections. All three traces contain the full payloads of all packets. The ICSI and ICSI2 traces only contain inbound traffic to TCP port 80, and are IP-source-anonymized.

Finally, Newsome et. al. [132] presented Polygraph, a signature generation system that is able to detect polymorphic worms by monitoring the network traffic. Polygraph generates signatures that consist of multiple disjoint content sub-strings. In doing so, Polygraph leverages the insight that for a real-world exploit to function properly, multiple invariant substrings must often be present in all variants of a payload; these substrings typically correspond to protocol framing, return addresses, and in some cases, poorly obfuscated code. They used several network traces as input for and to evaluate Polygraph signature generation.

For their HTTP experiments, the authors used two traces containing both incoming and outgoing requests, taken from the perimeter of Intel Research Pittsburgh in October of 2004. They used a 5-day trace (45,111 flows) as their innocuous HTTP pool and a 10-day trace (125,301 flows), taken 10 days after the end of the first trace, as an evaluation trace. The evaluation trace was used to measure the false positive rate of generated signatures. In experiments with noisy suspicious pools, noise flows were drawn uniformly at random from the evaluation pool. In addition, the authors also used a 24-hour DNS trace, taken from a DNS server that serves a major academic institutional domain. In particular, the first 500,000 flows from this trace was used as innocuous DNS pool, and the last 1,000,000 flows as evaluation trace.

5.3 Malware Detection Through DNS Analysis

The Domain Name System (DNS) has been increasingly being used by attackers to maintain and manage their malicious infrastructures. As a result, recent research on botnet detection has proposed number of approaches that leverage the distinguishing features between malicious and benign DNS usage.

The first study [171] in this direction proposed to collect real-world DNS data for analyzing malicious behavior. The results of the passive DNS analysis showed that malicious domains that are used in Fast-Flux networks exhibit behavior that is different than benign domains. Similarly, Zdrnja et al. [177] performed passive monitoring to identify DNS anomalies. They collected the DNS traffic of University of Auckland between 15 May 2006 and 15th of January 2007. The data consists of 260 GB of raw DNS data (uncompressed pcap files) and 50 million DNS records in the database.

In general, botnet detection through DNS analysis follows two lines of research: The first line of research tries to detect domains that are involved in malicious activities. The goal is to identify infected hosts by monitoring the DNS traffic. The second line of research focuses on the behaviors of groups of machines in order to determine if they are infected (e.g., a collection of computers always contact the same domain repeatedly).

5.3.1 Identifying Malicious Domains

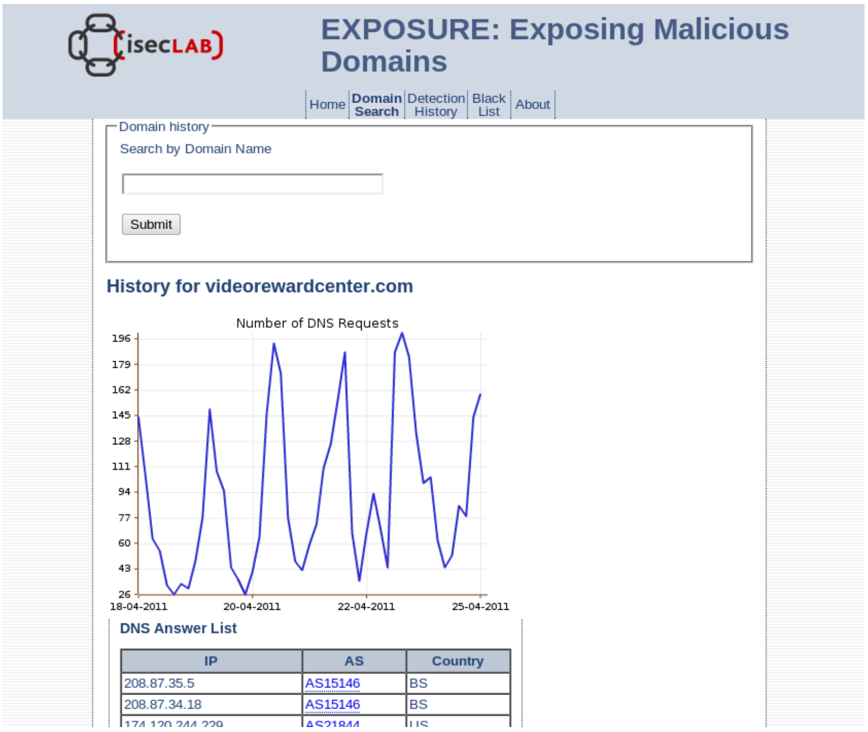
To detect malicious domains, previous approaches make use of passive DNS analysis, active DNS probing, and WHOIS [36] information. For example, recent work by Perdisci et al. [140] performs passive DNS analysis on recursive DNS traffic collected from number a number of ISP networks with the aim of detecting malicious Fast-Flux services. Contrary to the previous work [111, 131, 139, 165], Perdisci's work does not rely on analyzing blacklisted domains, and domains that are extracted from spam mails.

Perdisci et al. placed two traffic sensors in front of two different RDNS servers of a large north American Internet Ser- vice Provider (ISP). These two sensors monitored the RDNS traffic coming from users located in the north- eastern and north-central United States, respectively. Overall, the sensors monitored the live RDNS traffic generated by more than 4 million users for a period of 45 days, between March 1 and April 14, 2009. During this period, they observed an average of about 1.3 billion DNS queries of type A and CNAME per sensor. Overall they monitored over 2.5 billion DNS queries per day related to hundreds of millions of distinct domain names.

Another type of study on detecting malicious domains leverages properties inherent to domain registrations and their appearance in DNS zone files [85]. That is, they associate the registration information and DNS zone properties of domains with the properties of known blacklisted domains for proactive domain blacklisting. This method completely relies on historical information. Therefore, it is not able to detect domains that do not have any registration information and DNS zone commonalities with known blacklisted domains.

Generic Identification of Malicious Domains Using Passive DNS Monitoring

To date, two systems have been proposed for detecting malicious domains using passive DNS analysis: Notos [47] and EXPOSURE [67]. Notos dynamically assigns reputation scores to domain names whose maliciousness has not been discovered yet. The premise of this system is that malicious, agile use of DNS has unique characteristics and can be distinguished from legitimate, professionally provisioned DNS services. Notos uses passive DNS query data and analyzes the network and zone features of domains. It



EXPOSURE: Exposing Malicious Domains

Domain history

Search by Domain Name

History for videorewardcenter.com

DNS Answer List

IP	AS	Country
208.87.35.5	AS15146	BS
208.87.34.18	AS15146	BS
174.120.244.229	AS21844	US

Figure 5.1: Exposure Online Service

builds models of known legitimate domains and malicious domains, and uses these models to compute a reputation score for a new domain indicative of whether the domain is malicious or legitimate.

Notos uses the DNS traffic from two ISP-based sensors, one located on the US east coast (Atlanta) and one located on the US west coast (San Jose). Additionally it uses the aggregated DNS traffic from the different networks covered by the SIE [41]. In total, the database collected 27,377,461 unique resolutions from all these sources over a period of 68 days, from 19th of July 2009 to 24th September 2009.

EXPOSURE [67] eliminates several shortcomings of Notos. It does not require a wide overview of malicious activities on the Internet, a much shorter training time, and is able to classify domains that Notos would miss-classify. They introduce a passive DNS analysis approach and a detection system to effectively and efficiently detect domain names that are involved in malicious activity. They use 15 features that allow us to characterize different properties of DNS names and the ways that they are queried. In their experiments, they analyzed the DNS feeds provided by SIE@ISC [41] and the DNS traffic produced a by a network of 30000 clients.

During the two and a half month offline experimental period, EXPOSURE recorded and then analyzed 4.8 million distinct domain names that were queried by real Internet users. For the real-time detection experiments, they collected two-weeks of DNS data that consists of 100 million DNS queries. Unfortunately, tracking, recording and post-processing this volume of traffic without applying any filtering was not feasible in practice. Hence, they reduced the volume of traffic to a more manageable size by using two filtering policies. The goal of these policies was to eliminate as many queries as possible that were not relevant for them. However, they also had to make sure that they did not miss relevant, malicious domains.

The first policy they used whitelisted popular, well-known domains that were very unlikely to be malicious. To create this whitelist, they used the Alexa Top 1000 Global Sites [39] list. Their premise was that the most popular 1000 websites on the Internet would not likely be associated with domains that were involved in malicious activity. These sites typically attract many users, and are well-maintained and monitored. Hence, a malicious popular domain cannot hide its malicious activities for long. Therefore, they did not record the queries targeting the domains in this whitelist. The domains in the whitelist received 20 billion queries during two and a half months. By applying this first filtering policy, they were able to reduce 20% of the traffic they were observing.

The second filtering policy targeted domains that were older than one year. The reasoning behind this policy was that many malicious domains are disclosed after a short period of activity, and are blacklisted. As a result, some miscreants have resorted to using domain generation algorithms (DGA) to make it more difficult for the authorities to blacklist their domains. For example, well-known botnets such as Mebroot and Conficker [164] deploy such algorithms for connecting to their command and control servers. Typically, the domains that are generated by DGAs and registered by the attackers are new domains that are at most several months old. In their data set, they found 45.000 domains that were older than one year. These domains received 40 billion queries. Hence, the second filtering policy reduced 50% of the remaining traffic, and made it manageable in practice.

Clearly, filtering out domains that do not satisfy their age requirements could mean that they may miss malicious domains for the training that are older than one year. However, their premise is that if a domain is older than one year and has not been detected by any malware analysis tool, it is not likely that the domain serves malicious activity. To verify the correctness of their assumption, they checked if they had filtered out any domains that were suspected to be malicious by malware analysis tools such as Anubis and Wepawet. Furthermore, they also queried reports produced by Alexa [39], McAfee Site Advisor [42], Google Safe Browsing [40] and Norton Safe Web [43]. 40 out of the 45, 000 filtered out domains (i.e., only 0.09%) were reported by these external sources to be risky or shady. They therefore believe that their filtering policy did not miss a significant number of malicious domains because of the pre-filtering their performed during the offline experiments.

Bibliography

[1] Adobe flash player multimedia file remote buffer overflow vulnerability. http://www.securityfocus.com/bid/28695.

[2] Advanced Honey Pot Identification and Exploitation. http://phrack.ru/63/p63-0x09.txt.

[3] Beagle worm. http://www.symantec.com/security_response/writeup.jsp?docid=2004-011815-3332-99.

[4] Capture-HPC. https://projects.honeynet.org/capture-hpc.

[5] Explaining the “don’t click” clickjacking tweetbomb.

[6] HoneyClient. http://www.honeyclient.org.

[7] Honeytrap. http://honeytrap.mwcollect.org/.

[8] LaBrea. http://labrea.sourceforge.net/labrea-info.html.

[9] Leurre.com honeypot project. http://www.leurrecom.org/.

[10] Litterbox. http://www.wiul.org.

[11] LMbench Performance Analysis Tool. http://www.bitmover.com/lmbench.

[12] Microsoft Security Bulletin MS04-028. http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx.

[13] Microsoft Security Bulletin MS06-001. http://www.microsoft.com/technet/security/bulletin/MS06-001.mspx.

[14] Multipot. http://labs.idefense.com/files/labs/releases/previews/multipot/index.html.

[15] MyDoom worm. http://www.symantec.com/security_response/writeup.jsp?docid=2004-012612-5422-99.

[16] Nepenthes. http://nepenthes.mwcollect.org/.

[17] Network of Affined Honeypots (NOAH). http://www.fp6-noah.org.

[18] Nmap. http://insecure.org/nmap/.

[19] p0f. http://lcamtuf.coredump.cx/p0f.shtml.

[20] Predict. https://www.predict.org/.

[21] Qemu, open source processor emulator. http://fabrice.bellard.free.fr/qemu/.

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

[23] Sebek homepage. *http://www.honeynet.org/tools/sebek/*.

[24] Shelia. http://www.cs.vu.nl/~herbertb/misc/shelia/.

[25] Snort intrusion detection/prevention system. http://www.snort.org/.

[26] The social engineer toolkit. http://www.social-engineer.org.

[27] Stealing mouse clicks for banner fraud.

[28] The norman Sandbox. http://sandbox.norman.no.

[29] The Protocol Informatics Project. http://www.4tphi.net/~awalters/PI/PI.html.

[30] The Reusable Unknown Malware Analysis Net. http://www.secureworks.com/.

[31] User-mode Linux Kernel. http://user-mode-linux.sourceforge.net/.

[32] Using honeyclients to Detect New Attacks. http://www.synacklabs.net/honeyclient/Wang-Honeyclient-ToorCon2005.pdf.

[33] Virus Total. http://www.virustotal.com/.

[34] VMware. http://www.vmware.com.

[35] Worldwide Observatory of Malicious Behaviors and Attack Threats . http://wombat-project.eu/.

[36] RFC1834 - Whois and Network Information Lookup Service, Whois++. http://www.faqs.org/rfcs/rfc1834.html, 1995.

[37] Honeynet Project, Know Your Enemy: GenII Honeynets . http://www.honeynet.org/papers/gen2/index.html, May 2005.

[38] An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 375–388, New York, NY, USA, 2007. ACM.

[39] Alexa Web Information Company. http://www.alexa.com/topsites/, 2009.

[40] Google Safe Browsing. http://www.google.com/tools/firefox/safebrowsing/, 2010.

[41] Internet Systems Consortium. https://sie.isc.org/, 2010.

[42] McAfee SiteAdvisor. http://www.siteadvisor.com/, 2010.

[43] Norton Safe Web. http://safeweb.norton.com/, 2010.

[44] *BADGERS '11: Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, New York, NY, USA, 2011. ACM.

[45] J.M. Agosta, C. Diuk-Wasser, J. Chandrashekar, and C. Livadas. An adaptive anomaly detector for worm detection. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, page 3. USENIX Association, 2007.

[46] K.G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E.P. Markatos, and A.D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14^{th} Usenix Security Symposium*, August 2005.

[47] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *19th Usenix Security Symposium*, 2010.

[48] Spiros Antonatos, Iasonas Polakis, Thanasis Petsas, and Evangelos P. Markatos. A systematic characterization of im threats using honeypots. In *NDSS*, 2010.

[49] Vinod Anupam, Alain Mayer, Kobbi Nissim, Benny Pinkas, and Michael K. Reiter. On the security of pay-per-click and other web advertising schemes. *Comput. Netw.*, 31:1091–1100, May 1999.

[50] Gildas Avoine. Fraud within asymmetric multi-hop cellular networks. In Andrew S. Patrick and Moti Yung, editors, *Financial Cryptography*, volume 3570 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005.

[51] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. In *Proceedings of The 12th Annual Network and Distributed System Security Symposium (NDSS)*, February 2005.

[52] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, D. Watson, et al. The Internet Motion Sensor: A distributed blackhole monitoring system. In *Proceedings of the 12th ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 167–179. Citeseer, 2005.

[53] 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.

[54] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jaha-nian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID07)*, 2007.

[55] Michael Bailey, Evan Cooke, David Watson, Farnam Jahanian, and Niels Provos. A hybrid honeypot architecture for scalable network monitoring. Technical report cse-tr-499-04, Department of Electrical Engineering, University of Michigan, October 2004.

[56] Vijay A. Balasubramaniyan, Aamir Poonawalla, Mustaque Ahamad, Michael T. Hunter, and Patrick Traynor. PinDr0p: using single-ended audio features to determine call provenance. In *CCS '10: Proceedings of the 17th ACM conference on Computer and communications security*, pages 109–120, New York, NY, USA, 2010. ACM.

[57] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A solution for the automated detection of clickjacking attacks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 135–144, New York, NY, USA, 2010. ACM.

[58] Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Efficient Detection of Split Personalities in Malware. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, 2010.

[59] 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)*, October 2003.

[60] U. Bayer. Anubis: Analyzing Unknown Binaries. `http://analysis.seclab.tuwien.ac.at/`.

[61] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *16th Symposium on Network and Distributed System Security (NDSS)*, 2009.

[62] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. Insights Into Current Malware Behavior, 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[63] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A View on Current Malware Behaviors. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[64] Ulrich Bayer, Andreas Moser, Christopher Kruegel, , and Engin Kirda. Dynamic Analysis of Malicious Code. In *Journal in Computer Virology, Springer Computer Science*, 2009.

[65] V.H. Berk, R.S. Gray, and G. Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proceedings of the SPIE AeroSense*, volume 2, 2003.

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

[67] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *18th Annual Network and Distributed System Security Symposium, (NDSS 2011), San Diego*, February 2011.

[68] J. Binkley and S. Singh. An Algorithm for Anomaly-based Botnet Detection. In *Usenix Steps to Reduce Unwanted Traffic on the Internet (SRUTI)*, 2006.

[69] Jeffrey Carr. *Inside Cyber Warfare: Mapping the Cyber Underworld*. O'Reilly Media, Inc., 1st edition, 2009.

[70] Xu Chen, Jon Andersen, Z. Morley Mao, Michael Bailey, and Jose Nazario. Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware. In *Proceedings of the 38th Annual IEEE International Conference on Dependable Systems and Networks (DSN)*, 2008.

[71] Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights from the inside: a view of botnet management from infiltration. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[72] Marco Cova, Corrado Leita, Olivier Thonnard, Angelos D. Keromytis, and Marc Dacier. An analysis of rogue av campaigns. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 442–463, Berlin, Heidelberg, 2010. Springer-Verlag.

[73] 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.

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

[75] T. Cymru. The underground economy: priceless. http://www.usenix.org/publications/login/2006-12/openpdfs/cymru.pdf, December 2006.

[76] The Team Cymru. The team cymru darknet project. June 2004.

[77] 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, October 2004.

[78] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. Honeystat: Local worm detection using honeypots. In *Recent Advances in Intrusion Detection*, pages 39–58. Springer, 2004.

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

[80] M. Dornseif, T. Holz, and C. Klein. Nosebreak - attacking honeynets. In *Proceedings of the 5th IEEE Information Assurance Workshop*, June 2004.

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

[82] Joris Evers. Phishers hijack im accounts. http://news.cnet.com/Phishers-hijack-IM-accounts/2100-7349_3-6126367.html, October 2006.

[83] Hanno Fallmann, Gilbert Wondracek, and Christian Platzer. Covertly probing underground economy marketplaces. In *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'10, pages 101–110, Berlin, Heidelberg, 2010. Springer-Verlag.

[84] Hanno Fallmann, Gilbert Wondracek, and Christian Platzer. What's clicking what? techniques and innovations of today's clickbots. In *Proceedings of the 8th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'11, Berlin, Heidelberg, 2011. Springer-Verlag.

[85] Mark Felegyhazi, Christian Kreibich, and Vern Paxson. On the potential of proactive domain blacklisting. In *Proceedings of the Third USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, San Jose, CA, USA, April 2010.

[86] Peter Ferrie. Attacks on Virtual Machine Emulators. Technical report, Symantec Research White Paper, 2006.

[87] Peter Ferrie. Attacks on More Virtual Machines, 2007.

[88] Mona Gandhi, Markus Jakobsson, and Jacob Ratkiewicz. Badvertisements: Stealthy click-fraud with unwitting accessories. In *Online Fraud, Part I Journal of Digital Forensic Practice, Volume 1, Special Issue 2*, page 2006, 2006.

[89] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y. Zhao. Detecting and characterizing social spam campaigns. In *IMC '10*, pages 35–47, New York, NY, USA, 2010. ACM.

[90] M. Gheorghescu. An Automated Virus Classi?cation System. In *Virus Bulletin conference*, 2005.

[91] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *Workshop on Hot Topics in Understanding Botnets*, 2007.

[92] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. @spam: the underground on 140 characters or less. In *Proc. of the 17th ACM conf. on Computer and Communications Security*, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[93] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Usenix Security Symposium*, 2008.

[94] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *16th Usenix Security Symposium*, 2007.

[95] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[96] 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, April 2006.

[97] Mark Hofman. There is some smishing going on in the eu. http://isc.sans.org/diary.html?storyid=6076, March 2009.

[98] T. Holz, C. Willems, K. Rieck, P. Duessel, , and P. Laskov. Learning and Classification of Malware Behavior. In *Proceedings of the Fifth Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 08)*, 2008.

[99] Galen Hunt and Doug Brubacher. Detours: binary interception of win32 functions. In *Proceedings of the 3rd conference on USENIX Windows NT Symposium - Volume 3*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[100] Internet Identity (IID). Phishing trends report: First quarter 2010. Technical report, 2010.

[101] R. James, Z. Diego, and D. Yann. Building and deploying billy goat, a worm detection system. In *Proceedings of the 18th Annual FIRST Conference*, 2006.

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

[103] Noah M. Johnson, Juan Caballero, Kevin Zhijie Chen, Stephen McCamant, Pongsin Poosankam, Daniel Reynaud, and Dawn Song. Differential Slicing: Identifying Causal Execution Differences for Security Applications. In *IEEE Symposium on Security and Privacy (2011) (to appear)*.

[104] Min Gyung Kang, Heng Yin, Steve Hanna, Steve McCamant, and Dawn Song. Emulating Emulation-Resistant Malware. In *Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec)*, 2009.

[105] Alexandros Kapravelos, Iasonas Polakis, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos. D(e—i)aling with voip: Robust prevention of dial attacks. In *ESORICS*, pages 663–678, 2010.

[106] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale Botnet Detection and Characterization. In *Usenix Workshop on Hot Topics in Understanding Botnets*, 2007.

[107] Timo Kasper, Michael Silbermann, and Christof Paar. All you can eat or breaking a real-world contactless payment system. In *Financial Cryptography*, pages 343–350, 2010.

[108] H.A. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *13th USENIX Security Symposium*, pages 271–286, August 2004.

[109] 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, August 2000.

[110] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. In *J. Mach. Learn. Res,7:27212744*, 2006.

[111] M. Konte, N. Feamster, and J. Jung. Dynamics of online scam hosting infrastructure. In *In Passive and Active Measurement Conference*, 2009.

[112] Munir Kotadia. Phishers target yahoo instant messenger. http://www.zdnet.com.au/news/security/soa/Phishers-target-Yahoo-Instant-Messenger/0,130061744,139185847,00.htm, March 2005.

[113] Brian Krebs. Massive profits fueling rogue antivirus market. MassiveProfitsFuelingRogueAntivirusMarket, March 2009.

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

[115] Christian Kreibich, Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. On the spam campaign trail. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1:1–1:9, Berkeley, CA, USA, 2008. USENIX Association.

[116] Christian Kreibich, Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamcraft: an inside look at spam campaign orchestration. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'09, pages 4–4, Berkeley, CA, USA, 2009. USENIX Association.

[117] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *Recent Advances in Intrusion Detection*, pages 207–226. Springer, 2006.

[118] Nir Kshetri. The economics of click fraud. *IEEE Security and Privacy*, 8:45–53, May 2010.

[119] Boris Lau and Vanja Svajcer. Measuring virtual machine detection in malware using DSD tracer. *Journal in Computer Virology*, 6(3), 2010.

[120] Tobias Lauinger, Veikko Pankakoski, Davide Balzarotti, and Engin Kirda. Honeybot, your man in the middle for automated social engineering. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[121] T. Lee and J. J. Mody. Behavioral Classifcation. In *EICAR Conference*, 2006.

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

[123] Federico Maggi. Are the con artists back? a preliminary analysis of modern phone frauds. In *Proc. of the 10th IEEE Intl. Conf. on Computer and Information Technology*, pages 824–831, 2010.

[124] Federico Maggi, Alessandro Sisto, and Stefano Zanero. A social-engineering-centric data collection initiative to study phishing. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, BADGERS '11, pages 107–108, New York, NY, USA, 2011. ACM.

[125] D. Kevin McGrath and Minaxi Gupta. Behind phishing: an examination of phisher modi operandi. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 4:1–4:8, Berkeley, CA, USA, 2008. USENIX Association.

[126] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Detectives: detecting coalition hit inflation attacks in advertising networks streams. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 241–250, New York, NY, USA, 2007. ACM.

[127] Kevin D. Mitnick and William L. Simon. *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[128] Tyler Moore and Richard Clayton. Financial cryptography and data security. chapter Evaluating the Wisdom of Crowds in Assessing Phishing Websites, pages 16–30. Springer-Verlag, Berlin, Heidelberg, 2008.

[129] Tyler Moore, Richard Clayton, and Ross Anderson. The economics of online crime. *Journal of Economic Perspectives*, 23(3):3–20, 2009.

[130] Bob Mungamuru and Stephen Weis. Financial cryptography and data security. chapter Competition and Fraud in Online Advertising Markets, pages 187–191. Springer-Verlag, Berlin, Heidelberg, 2008.

[131] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *International Conference on Malicious and Unwanted Software*, 2008.

[132] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *IEEE Symposium on Security and Privacy*, pages 226–241, 2005.

[133] 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.

[134] Hamish O'Dea. The modern rogue - malware with a face. In *Virus Bulletin Conference 2009*, 2009. http://www.microsoft.com/downloads/en/details.aspx?FamilyID=7A827FBD-C2A1-48BC-9E85-6B805D3E7E26.

[135] Gunter Ollmann. The vishing guide. Technical report, IBM, May 2007.

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

[137] Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. In *Proceedings of the 3rd USENIX Workshop on Offensive Technologies (WOOT)*, 2009.

[138] 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.

[139] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. Fluxor: Detecting and monitoring fast-flux service networks. In *Detection of Intrusions and Malware, and Vunerability Assessment*, 2008.

[140] R. Perdischi, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *25th Annual Computer Security Applications Conference (ACSAC)*, 2009.

[141] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G.M. Voelker, V. Paxson, N. Weaver, and S. Savage. Botnet Judo: Fighting Spam with Itself . In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, March 2010.

[142] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proceedings of ACM SIGOPS Eurosys 2006*, April 2006.

[143] 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.

[144] Niels Provos. A virtual honeypot framework. In *Proceedings of the 12th USENIX Security Symposium*, pages 1–14, August 2003.

[145] Siles R. Honeyspot: The wireless honeypot. The Spanish Honeynet Project http://honeynet.org.es/papers/honeyspot/HoneySpot_20071217.pdf, 2007.

[146] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. Detecting System Emulators. In *Information Security Conference (ISC)*, 2007.

[147] M.A. Rajab, F. Monrose, and A. Terzis. On the Effectiveness of Distributed Worm Monitoring.

[148] Moheeb Abu Rajab, Lucas Ballard, Panayiotis Mavrommatis, Niels Provos, and Xin Zhao. The nocebo effect on the web: an analysis of fake anti-virus distribution. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[149] M. Reiter and T. Yen. Traffic aggregation for malware detection. In *DIMVA*, 2008.

[150] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *13th Systems Administration Conference (LISA)*, 1999.

[151] Joanna Rutkowska. Red Pill... or how to detect VMM using (almost) one CPU instruction. http://invisiblethings.org/papers/redpill.html, 2004.

[152] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, 2010.

[153] Jimmy Shah. School of smish. http://www.avertlabs.com/research/blog/?p=75, August 2006.

[154] Youngsang Shin, Minaxi Gupta, and Steven Myers. The nuts and bolts of a forum spam automator. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, LEET'11, pages 3–3, Berkeley, CA, USA, 2011. USENIX Association.

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

[156] S. Singh, C. Estan, G. Varghese, and S. Savage. The earlybird system for real-time detection of unknown worms. Technical report, Citeseer, 2003.

[157] 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)*, February 2007.

[158] A.B. Smith and J.M. Fox. RandomNet. In *http://www.citi.umich.edu/u/provos/honeyd/ch01-results/3/*, March 2003.

[159] Brett Stone-Gross, Ryan Abman, Christopher Kruegel Richard Kemmerer, Douglas Steigerwald, and Giovanni Vigna. The underground economy of fake antivirus software. In *WEIS*, 2011.

[160] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[161] Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The underground economy of spam: a botmaster's perspective of coordinating large-scale spam campaigns. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, LEET'11, pages 4–4, Berkeley, CA, USA, 2011. USENIX Association.

[162] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting Botnets with Tight Command and Control. In *31st IEEE Conference on Local Computer Networks (LCN)*, 2006.

[163] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 1–9, New York, NY, USA, 2010. ACM.

[164] The Telegraph. French fighter planes grounded by computer virus. http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html, 2010.

[165] T.Holz, C. Gorecki, K. Rieck, and F.C. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

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

[167] S. Venkataraman, D. Song, P.B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *Proc. NDSS*, pages 149–166. Citeseer, 2005.

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

[169] 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, February 2006.

[170] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 3–3. USENIX Association, 2004.

[171] F. Weimer. Passive DNS Replication. In *FIRST Conference on Computer Security Incident*, 2005.

[172] D. Whyte, E. Kranakis, and P. Van Oorschot. DNS-based detection of scanning worms in an enterprise network. In *Proc. of the 12th annual Network and Distributed System Security symposium*. Citeseer, 2005.

[173] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5:32–39, 2007.

[174] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically Generating Models for Botnet Detection. In *14th European Symposium on Research in Computer Security(ESORICS 2009)*, 2009.

[175] Guanhua Yan, Stephan Eidenbenz, and Emanuele Galli. Sms-watchdog: Profiling social behaviors of sms users for anomaly detection. In *RAID*, pages 202–223, 2009.

[176] 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.

[177] B. Zdrnja, N. Brownlee, and D. Wessels. Passive Monitoring of DNS anomalies. In *DIMVA*, 2007.

[178] Linfeng Zhang and Yong Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ICDCS '08, pages 77–84, Washington, DC, USA, 2008. IEEE Computer Society.

[179] Ruishan Zhang, Xinyuan Wang, Xiaohui Yang, and Xuxian Jiang. Billing attacks on sip-based voip systems. In *Proceedings of the first USENIX workshop on Offensive Technologies*, pages 4:1–4:8, Berkeley, CA, USA, 2007. USENIX Association.

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