

HAKING EXTRA

Issue 1/2012 (8) ISSN 1733-7186

HONEYPOTS

THE SITTING DUCK ON THE NETWORK



**PROACTIVE NETWORK DEFENCE THROUGH
SIMULATED NETWORKS**

**THE GAME OF GIVING MALWARE A NAME
USING HONEYCLIENTS TO DETECT MALWARE**

PLUS

**EXCLUSIVE INTERVIEW
WITH FRED COHEN**

ashampoo®

Szukaj nas także na



www.ashampoo.com

The Industry's First Commercial Pentesting Drop Box.

THE **Pwn Plus.**



Air Freshener?

Printer PSU?
...nope

FEATURES:

- ★ Covert tunneling
- ★ SSH access over 3G/GSM cell networks
- ★ NAC/802.1x bypass
- ★ and more!



PWNIE EXPRESS

@ **pwnieexpress.com**

t) @pwnieexpress **e)** info@pwnieexpress.com **p)** 802.227.2PWN

Discover the glory of
Universal Plug & Pwn

**Managing:**

Michał Wiśniewski
m.wisniewski@software.com.pl

Senior Consultant/Publisher:

Paweł Marciniak

Editor in Chief:

Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Art Director:

Marcin Ziółkowski

DTP:

Marcin Ziółkowski
www.gdstudio.pl

Production Director:

Andrzej Kuca
andrzej.kuca@hakin9.org

Marketing Director:

Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Proofreaders:

Bob Folden, I. Motz, Michał Wiśniewski

Top Betatesters:

Tom Updegrove, Lou Lombardy,
Rissone Ruggero, Ismael Valenzuela, Eder Lira,
Gerardo Iglesias Galvan, Kiran Vangaveti,
Daniel Sligar, David von Vistauxx, I. Motz,
Tahir Saleem, Bert White, Rissone Rugero,
Marek Janac, Tyler Hudakville, Ayo Tayo-
Balogun, Patrik Gange, Massimiliane Sembiente,
Kapo Li, Dan Dieterle, Nana Onumah,
Jeff Smith, Tim Thorniley, Jonathan Ringler,

Publisher: Software Media Sp. z o.o. SK
02-682 Warszawa, ul. Bokserska 1
www.hakin9.org/en

Whilst every effort has been made to ensure the high quality of the magazine, the editors make no warranty, express or implied, concerning the results of content usage. All trade marks presented in the magazine were used only for informative purposes. All rights to trade marks presented in the magazine are reserved by the companies which own them.

To create graphs and diagrams we used program by Mathematical formulas created by Design Science MathType™ **DISCLAIMER!**

The techniques described in our articles may only be used in private, local networks. The editors hold no responsibility for misuse of the presented techniques or consequent data loss.

TO HACK OR NOT TO HACK – THAT IS THE QUESTION

DEAR HAKIN9 EXTRA FOLLOWERS, WE ARE GIVING YOU THE LATEST FRUIT OF OUR LABOUR. HONEYPOTS ARE OUR LE-ITMOTIV THIS MONTH. ESPECIALLY FOR YOU, OUR DEAR FOLLOWERS, WE HAVE SELECTED THE CHOICEST ARTICLES WITHIN THE TOPIC OF HONEYPOTS/HONEYNETS. I SINCERELY HOPE THAT WE SUFFICIENTLY EXPANDED ON THE TOPIC TO SATISFY YOUR NEEDS AND WE QUENCHED YOUR APPETITE FOR HAKIN9 KNOWLEDGE. I AM ALSO VERY HAPPY THAT WE MANAGED TO HAVE AN EXCLUSIVE INTERVIEW WITH DR. FRED COHEN – THE „FATHER” OF COMPUTER VIRUSES AND THAT, ONCE AGAIN, OUR RESPECTED AUTHORS HELPED US WITH THEIR CONTRIBUTIONS. THIS MONTH: JEREMIAH BROTT WILL, IN GREAT DETAIL, TELL YOU ABOUT DIFFERENT TYPES OF HONEYPOTS AND THEIR USE. ROBERTO SAIA IS GOING TO PRESENT YOU „PROACTIVE NETWORK DEFENCE THROUGH SIMULATED NETWORKS”. HARI KOSARAJU’S ARTICLE WILL EXPOSE HONEYPOT AS A SYSTEM WHICH SOLE PURPOSE IS TO BE ATTACKED AND AS FRED COHEN STATED: „ONE PERSON’S ATTACK IS ANOTHER PERSON’S INTELLIGENCE OPERATION.” CLIENT HONEYPOTS, THEIR USE AND INTER-ACTIONS ARE DESCRIBED IN THE ARTICLE WRITTEN BY MIROSLAV LUDVIK AND MICHAL SRNEC. DAVIDE CANALI WILL SHOW YOU HOW TO USE HONEYCLIENTS IN MALWARE DETECTION. SPEAKING OF MALWARE, MICHAEL BOELEN WILL PRESENT YOU THE STORIES BEHIND GIVING NAME TO MALWARE. ARE YOU SATISFIED NOW? IF NOT, ESPECIALLY FOR YOU DEAR READERS WE HAVE AN INTERVIEW WITH FRED COHEN. DR. COHEN AGREED TO TALK ABOUT NETWORK DEFENCE, HIS BEGINNINGS, HONEYPOTS, FAVOURITE S-F TV SERIES AND MANY MORE. CHECK US OUT BY GRABBING HAKIN9 EXTRA.

ON BEHALF OF HAKIN9 EXTRA I WOULD LIKE TO WISH YOU: HAPPY VIOLENTINES, HAKIN9 VALENTINES OR SIMPLY..HAPPY VALENTINES. WE HOPE THAT AFTER THIS ISSUE YOU WILL LOVE HAKIN9 EXTRA EVEN MORE (HOW CAN ONE LOVE HAKIN9 EXTRA MORE?) AND FOLLOW US IN EVERY STEP.

STAY TUNED!!!

MICHAŁ, HAKIN9 EXTRA

EaseUS® Todo Backup Advanced Server 4.0

Top Reliable Backup Solution for Business

Secure

Save backups to a safe place, with the protection of password

Fast and Reliable

Top Reliable Backup Solution for Business

Guaranteed

Verify the integrity of backups for reliable recovery



Easy-to-use

Non-specialists can perform backup and recovery

Why choose EaseUS Todo Backup Advanced Server?

It is the Windows Server backup and disaster recovery solution for both physical and virtual Windows Server environments. It provides powerful server data backup & recovery features such as fast system & files backup, disaster recovery, Exchange/SQL Server backup & recovery, advanced backup scheduler and one-click to restore system to dissimilar hardware. It will simplify both basic and advanced server backup & recovery management while minimizing server downtime as well as ensuring business continuity.

8. Honey Pots – the Sitting Duck on the Network

by Jeremiah Brott

The purpose of this article is to provide details on what honey pots are, the characteristics of the two types down to the mechanics of how each one works. It will also analyze the benefits and pitfalls to explore multiple uses of a honey pot, from detection to prevention. It will also analyze some implementation techniques, design ideas and the possible legal issues surrounding them. Also explored is a honey pot specifically designed for malware analysis.

18. The Game of Giving Malware a Name

by Michael Boelen

While running a honeypot, it suddenly gets “infected” with a piece of malware. After a quick look, it seems this catch could be an unknown species. After validating it with several malware scanning tools, it seems no one has ever seen it, or created an appropriate signature for it yet. So is this really a new piece of malicious software or simply one of the utilities used by the intruder? The hunt to find the identity of this odd visitor is about to start...

22. Proactive Network Defence through Simulated Network

by Roberto Saia

A honeypot-based solution realizes a credible simulation of a complete network environment where we can add and activate one or more virtual hosts (the honeypots) in various configuration: a network of honeypot systems is named honeynet.

32. Using Honeypots to Strengthen Network Security

by Hari Kosaraju

Honeypots have emerged as a new class of network security technology to address some of the shortfalls of existing solutions. In this article, we will first discuss the limitations of current network threat detection technologies. Next we will introduce various classes of Honeypots and how they differ. Third, we will examine how a potential attacker could detect a Honeypot and then, we will learn how Honeypots can be used to detect Zero Day attacks. We will conclude by discussing cloud based Honeypot architectures.

38. Client Honeypots

by Michal Srnec and Miroslav Ludvik

Development of security tools has been on the rise in recent years. The main reason of that is the wide variety of attack trends against computer systems. This new technologies like intrusion detection systems (IDS), anti-viruses and firewall, help to address this issues. One of this new technologies is honeypot. Honeypot is pretty new technology which use different technique to help address security problems. One of the many definition of honeypot is *a resource whose value is in being probed, attacked or compromised*. Another resource define honeypots like: *A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource*. In other (user friendly) words: Underlying strategy is simple but really powerful – to allure potential attackers to fake network node and tracking the attacker operations. Based on this observation, system administrators can build the security policy.

42. Detecting Malware with Honeyclients

by Davide Canali

This article will first introduce you to the state of the art in the matter of malware detection using honeyclients, showing a short history of honeyclients and the different types of honeyclients on the market. Then, you'll learn how to setup one of the most recent and complete open source honeyclient systems, allowing you to analyze any kind of content (URLs, executable files, PDFs, documents, ...) on a virtual machine running Windows. In order to understand this article, you'll need only some basic knowledge of Linux and of the VirtualBox virtualization solution. A basic knowledge of Python is a plus, even though not necessary.

50. Exclusive Interview with Fred Cohen

by Nick Baronian

First off, protecting information is not the goal of information protection. The definition of protection is *keeping from harm*. That is, keeping people (and other creatures that feel pain and pleasure, live and die, etc.) from being harmed (information doesn't feel pain and is not harmed when altered). But harm associated with information...Information protection is a complex issue involving many equities. One person's attack is another person's intelligence operation. Is it protecting information in the form of financial records to not aggressively break into the systems of those who attack those same records? When you are attacked (whatever that may be) should you not be able to aggressively defend?

HONEY POTS

THE SITTING DUCK ON THE NETWORK

JEREMIAH BROTTW

There is an old saying that states in order to draw a good face you must first learn how to draw near perfect circles. After all, circles are the basic fundamental of drawing a face. Computer security follows this same suggestion. With the continuously evolving threats on the Internet, the basics must be covered first. Failure to learn the fundamental tools and techniques will result in the inability to draw the “perfect face”, in relation to computer security.



How can attacks be mitigated if they are not understood?

This is where a honey pot will come in to play. The purpose of this article is to provide details on what honey pots are, the characteristics of the two types down to the mechanics of how each one works. It will also analyze the benefits and pitfalls to explore multiple uses of a honey pot, from detection to prevention. It will also analyze some implementation techniques, design ideas and the possible legal issues surrounding them. Also explored is a honey pot specifically designed for malware analysis.

Honey pots are basically a system which has been designed to provide value by being attacked, probed or compromised. Unlike typical security devices, honey pots are designed to attract attackers, for the sole purpose of learning about an attacker(s) tools & techniques by closely interacting with them, while hopefully unbeknownst 'to the attacker(s)' logging details of the attack.

Using honey pots you can also learn a lot more about the tools & techniques being used by attacker(s) targeting your network, or information about the latest worm, malware, kiddie with a scanner, *<insert threat name here>* floating about on the internet.

Honey pots generally serve no production value from a corporate point of view, but they become invaluable when deployed properly from a computer security point of view. Some honey pots can be deployed for a purpose of attack prevention, while others are deployed for detection, information gathering or research purposes. Whatever type and deployment method you choose should be based on the results you wish to achieve from deploying such a tool.

If you plan on deploying a honey pot as a detection device, early warning system or as your network burglar alarm, then it is recommended to deploy a low-interaction based setup. See below.

If you plan on gathering extensive information on threats, 0 day vulnerabilities or learning more detailed information about the tools and techniques employed. Then you should be looking into a high-interaction based setup. See below.

Honey Pots – What are they?

A honey pot in the security world is known as a trap - it detects, deflects, or in some cases attempts to interact with the attacker(s). They are closely monitored machines that can function as a decoy. They can distract an attacker from more fruitful targets by appearing to be both vulnerable and important. They can also serve as an early warning alert system about new attacks and threats that are facing a network.

Since the purpose of a honey pot is to attract attackers, it is crucial that they are not deployed in a way that will allow them to interact with critical assets on a network.

The information obtained from running a honey pot can raise awareness about new attacks and trends, while also allowing people to gain insight into the attacker's methodology both during and after the exploitation.

Honey pots can be a highly flexible tool in your arsenal. Like other tools in computer security, they don't fix any one single problem. Security is a process, not a product. A honey pot is a useful tool for information gathering, prevention or detection. The primary role of a honey pot depends on how it is deployed. Despite all the differences and customization options available for various honey pot setups, they all share a common function – to be attacked and compromised.

Before deploying a honey pot, it is important to understand how they are classified. The classification is based on the amount of interaction between the attacker and the honey pot. There are generally two categories they are broken down into: Low Interaction and High Interaction.

Low Interaction

These setups will always have a limited interaction with attackers. As a consequence, there will only be limited information to obtain. The Low Interaction honey pots generally function by emulating a service on a specific operating system. Although the Low Interaction machines do not give as much information about an attack as the High

Interaction machines do, there are some specific advantages. The first advantage is that they are very easy to deploy and maintain. They also have a much lower risk level compared to High Interaction honey pots due to the nature of the service

being emulated. They could be considered the “plug-n-play” of the honey pot world.

Software

Deception Toolkit (open source) – ? DTK was the first open source honeypot to ever be released. Released in 1997 by Fred Cohen, DTK was a collection of Perl scripts and C source code that could emulate a variety of listening services. Its primary purpose was to deceive attackers. This tool is very dated and hard to find, but was well worth the mention since it was one of the first honey pots ever released.

Honeyd (open-source Linux) – *covered by this article* – <http://www.honeyd.org/> Honeyd is a small daemon that creates virtual hosts on a network. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. Honeyd enables a single host to claim multiple addresses.

mwcollect (open-source Linux) - *covered by this article* – <http://code.mwcollect.org/> mwcollectd v4, a next-generation low-interaction malware collection honey pot. It's written in C++, but the easy integration of additional Python modules means that malware researchers around the world can easily extend the honey pot with new protocols and features. Mwcollect was started using the best features of nepenthes and honey trap, licensed under the LGPL.

LaBrea Tarpit (open-source) – <http://labrea.sourceforge.net/> LaBrea is a unique honey pot, in that it is designed to slow down or stop attacks by acting as a 'sticky' honey pot. Also has the ability to run on multiple operating systems.

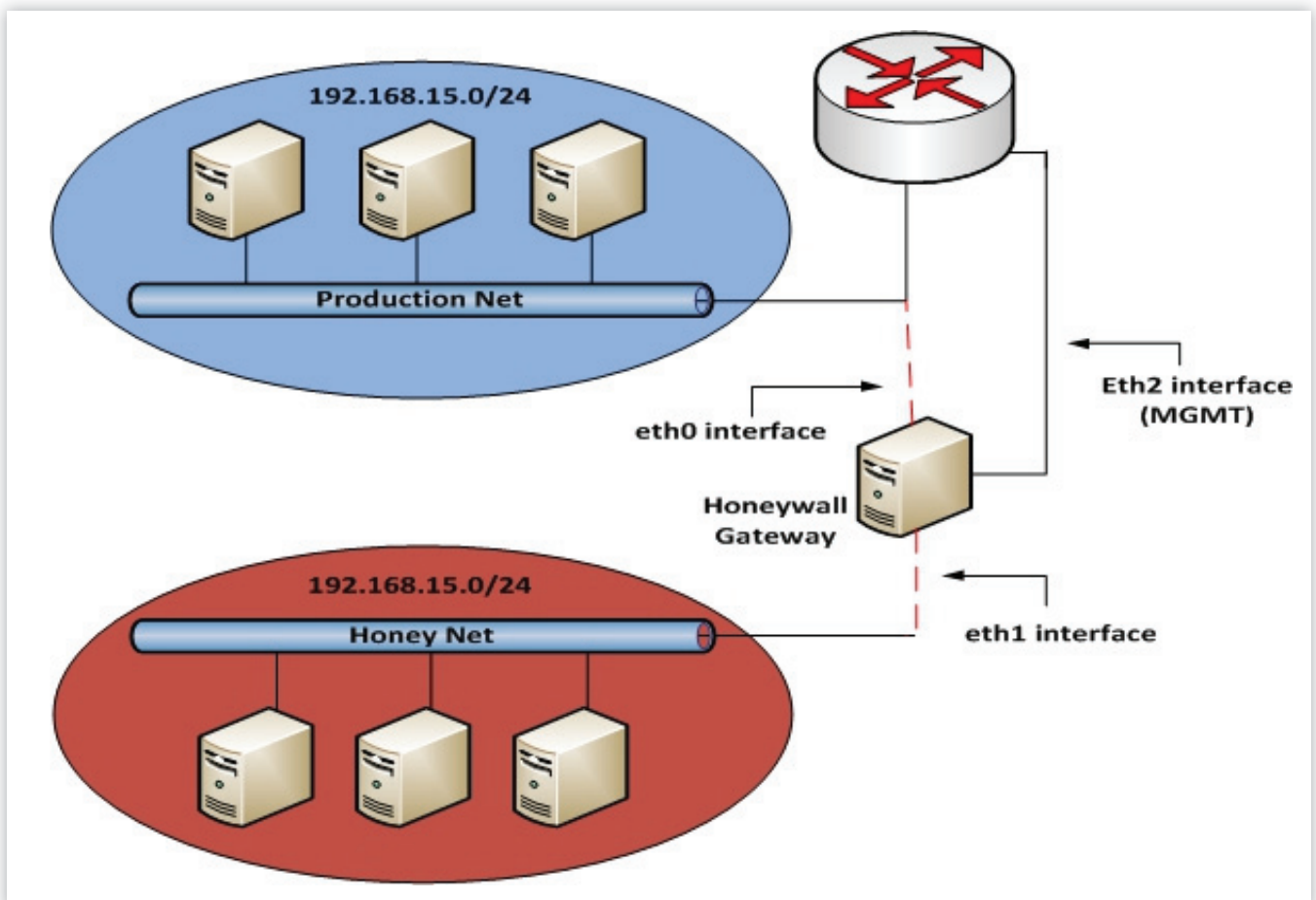


Figure 1.

KFSensor (commercial Windows) - <http://www.keyfocus.net/kfsensor/>.

KFSensor is a Windows based system that acts as a honey pot to attract and detect hackers and worms by simulating vulnerable system services and Trojans. By acting as a decoy server it can divert attacks from critical systems and provide a higher level of information than can be achieved by using firewalls and NIDS alone.

Specter (commercial Windows) - <http://www.specter.com>
Specter is a windows based low-interaction honey pot. It can emulate 13 different operating systems, monitor up to 14 ports, and comes loaded with configuration and notification options.

High Interaction

These setups are the other side of the duct tape. They are significantly more complex in their design and setup, as well as their overall maintenance. These honey pots do not emulate services at all. On the contrary, they employ actual services. This allows the honey pot's administrator to obtain much more detailed information concerning the attack.

They also have several advantages over low interaction honey pots. One of the advantages is that the honey pot makes no assumption about how an attacker will interact. Instead, they behave as though they were a host in a normal production environment. High interaction setups allow for the acquisition of extensive amounts of information, which is what gives the High interaction honey pots the biggest advantage over lower interaction. Since the services aren't emulated, the honey pot can capture unexpected behavior, or even information pertaining to malicious software, such as a root kit.

High Interaction honey pots are also more useful and comprehensive than low interaction honey pots, but they are often more costly and require external technology in order to deploy them properly.

Software

Honeywall CDROM (open-source LiveCD) - <http://www.honeynet.org/project/HoneywallCDROM> Honeywall is a CD ROM provided by the Honey net Project. This CD allows you to create architecture that allows you to deploy both low-interaction and high-interaction honey pots, but is mainly designed for high interaction deployments. The tools included allow for capturing, controlling and analyzing attacks.

Honey Nets

The primary focus on deploying high-interaction based honey pots within a honey net should always be on the bridging or firewall device that is separating the malicious honey net from the production network.

You can think of honey nets as a fish bowl in the pet store, but instead of fish with submarine diver inside, you have vulnerable Linux and Windows servers. Just as you would see fish interacting with these environments, attacker(s) will be interacting with your honey net environments.

The actual honey pots that live within the honey net should be various old operating systems, which are full of vulnerabilities just waiting to be attacked. The actual gateway or bridging device can be custom built or to save time and headaches it is recommended to use the Honeywall CDROM from the Honeynet Project.

Implementation and Design

There are four key requirements that should be used for a successful honey net implementation. These requirements are as follows: *Data Control, Data Capture, Data Analysis and Data Collection(optional)*.

Data control is the means of containing the activity within the honey net this is how you will mitigate the risk. Risk of running a honey pot comes into play as there is always a potential for an attacker to run some code that will enable them to either attack or otherwise harm other non-honey net related systems.

Data Capture is how you will monitor and log all of the threats activities within the honey net. Using this captured data will allow you to further analyze the information to learn about the tools, techniques and motives used by the attacker(s). The primary challenge faced with data capturing, is the ability to capture as much information as possible without alerting the attacker(s) of the capturing activity.

Data Analysis serves the third requirement for a successful implementation. Honey pots are all about information at the end of the day. Without the ability to convert the data collected from a honey pot into meaningful information essential renders a honey pot useless.

Data Collection comes in handy when you have deployed multiple honey nets across various locations, data collection is only useful when you wish to combine data for trending analysis.

While it all sounds daunting to implement, the fine folks at the Honeynet Project have cured this problem. The solution is a tool called Honeywall CDROM, the cdrom is a bootable environment which allows you to rapidly deploy a gateway device which implements all of the above requirements.

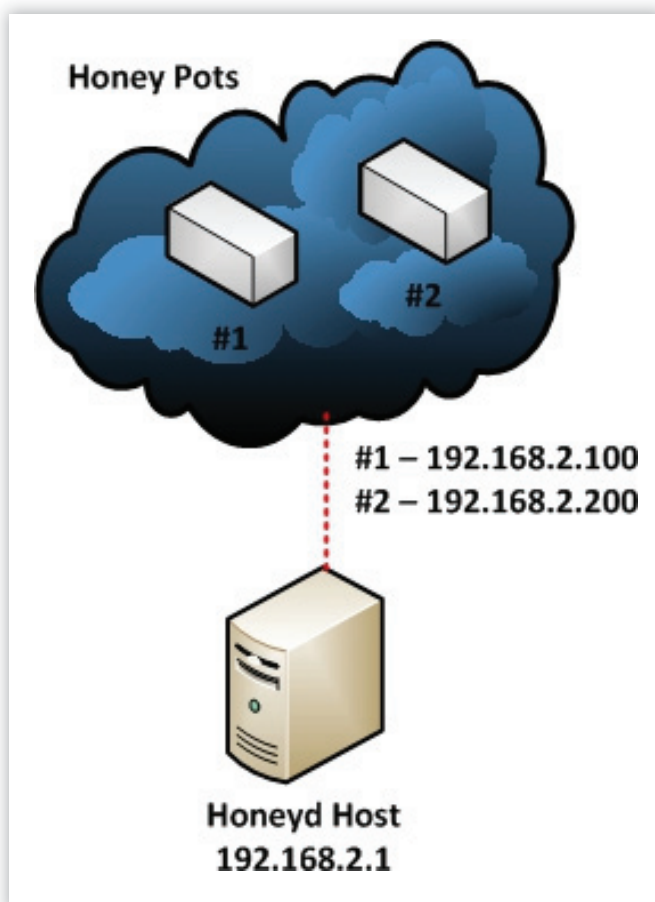


Figure 2.

Deployment Example

The diagram below is an example diagram of what a honey net architecture should look like. Using this architecture gives the ability to create a highly controlled network that can be closely monitored for malicious activities within it. In this setup, the Honeywall CDROM has been used.

The Honeywall CDROM acts as a bridge into and from the honey net. Using this method all traffic must pass through the Honeywall before entering or leaving the honey net. Since the Honeywall CDROM acts as a bridging device, the device should be invisible to anyone interacting with the honey pots.

The diagram below illustrates a honey net architecture. The honey pots in the deployment have been meshed within a production network, using the methods explained above.

In this setup the bridging device *Honeywall Gateway* has 3 interfaces. The first 2 interfaces (eth0 and eth1) will be used for the actual bridge. These interfaces will be what separates the honey net from the production network or everyone else on the network. These bridged interfaces simply act as a bump in the wire, meaning they have no IP stack. The third interface (eth2) is what is used for the management interface. Sometimes place on what is known as an OOBMN or Out Of Band Management Network.

Honey Pots continued...

In order to further classify honey pots, they can be broken down into two more sub-categories: production based and research based.

You will also hear the term, honey net. Honey nets are simple architecture designs. Meaning they are a network that contains more than one honey pot. Since honey nets are not production systems providing services, any interactions within the honey net its self implies malicious behaviour or un-authorized activities.

Commercial products such as TrustWave Mirage use similar techniques for the NAC based deployments. Any outbound activities from a honey net, automatically indicates evidence of a compromised system or malicious activities. Any inbound connections to the honey net will indicate signs of a scan or an incoming attack.

Deploying a honey net makes tracking and monitoring of malicious activity simple. Using IPS and firewall logs with correlation tools is still a daunting task for identifying attacks with you have terabytes of log data. Anything captured within a honey net is automatically assumed to be unauthorized and can be analyzed easier for malicious activities.

A production based setup is usually deployed using commercial or freely available open-source software, and are primarily used by a company or corporation. Production based honey pots are often found inside of an enterprise network, scattered amongst other production servers. This allows them to act like decoys in their environment.

In order for them to function as a decoy, the honey pot must assume that an attacker will go for easy targets first. These types of honey pots are considered to be Low interaction, since they are given a specific function or service to emulate within these environments. The purpose of these honey pots is to mitigate attacks, not necessarily research them.

A research honey pot is usually utilized by private security firms, "hobby" hackers, military and/or government, or perhaps just someone who is interested in the tactics and motives of hackers. These types of honey pots are incredibly useful for exposing the current threats that organizations face on a daily

basis. The information that is obtained by these honey pots is used to develop better ways to protect assets.

Understanding the value of a Honey pots

Some people may be wondering where the value is in deploying a honey pot. When you need to determine the value of the honey pot you are deploying, you first have to look at which sub category you will be using. Production honey pots can be deployed with a sole purpose of protection an organization by either preventing, detecting or acting as an early warning/alerting system. All of these will allow the organization to hopefully better respond to an attack before it hits a critical system.

When deploying a honey pot for a research or curiosity purpose, they are simply a tool to collect information. The value of the information depends on the intentions of the person running the honey pot. Organizations may deploy research based honey pots in order to aid law enforcement, or their own trending or tracking of malicious activities on the network.

Typically a high-interaction honey pot is deployed for research purposes, while a low-interaction honey pot would be deployed for production purposes. It usually all boils down to risk and threat levels. However it is up to the person(s) responsible for deploying and maintaining the honey pots as to which type will be used, both low and high interaction honey pots can serve a valid purpose in either research or production based deployments.

When honey pots are used within a production deployment, they will benefit the organization in one of three ways: prevention, detection and response.

A very well known example of using honey pots for a research purpose can be found on the Honey net Project website. <http://www.honeynet.org>

Deploying a Low-Interaction Honey pot

If this is your first experience with a honey pot, we advise against a high-interaction based setup. Start out using a low-interaction setup in order to get your feet wet.

If you're planning to deploy a honey net, you must first have a proper architecture in place to accommodate the honey pots safely away from other devices on your existing network. This is normally some sort of gateway/firewall device that segregates your honey pots from everyone else.

Any traffic going to or from the honey net MUST pass through this device. Using tools like the Honey net Honeywall CDROM, will simplify deploying a layer 2 bridging device that is invisible to anyone interacting with the honey pots that you have deployed.

The gateway device should contain a minimal of three (3) network interface cards. The first two (2) network cards will be used for segregating the honey pots from everything else on the network. These interfaces will act as a bridge between the two networks and contain no IP stack. The third interface has a valid IP stack which will allow for monitoring and administering the gateway. An ideal scenario this would be on what is known as an OOBMN, Out of Band Management Network.

The main requirements of the gateway can be met with implementing the following: Data Control, Data Capture, Data Analysis and Data Collection.

Data Control is how you can define the activity will be contained within the honey net without the attacker(s) noticing it.

Data Capture is the ability to capture ALL of the attacker(s) activities without being noticed. Data Analysis is the ability to analyze the data in real-time, while Data Collection allows you

to collect the data from multiple honey pots within you're honey net to a single/central source.

Guide Requirements

In order to follow along with this guide, several things will be required.

Debian 6 – (Virtual or Physical). This will be used for installing Honeyd & mwcollect.

NOTE: Virtual machine users will need to configure the ability for the VM to set promiscuous mode.

Honeyd

Honeyd is the BMW when it comes to low-interaction honey pots. First released in 2002 by Niels Provos, was written in C and designed for the UNIX platform. Honeyd is very unique as it has introduced multiple new concepts into the honey pot world. These including the ability to monitor millions of unallocated IP addresses, implements IP stack spoofing and can simulate 100s of different operating systems at the same time. It can also monitor any TCP or UDP port.

Setup Overview

Note: In order to follow along with this guide, ensure you have the following available.

Honeyd – Available via Debian repo

Attacker Machine (optional)

Main Host – Honeyd host.

IP Address: 192.168.2.1

Virtual Honey pot #1 - Microsoft Windows NT 4.0 SP3

IP Address: 192.168.2.100

Virtual Honey pot #2 - IBM AIX 4.2

IP Address: 192.168.2.200

Installation

To install Honeyd & supporting packages on Debian 6 is very simple. From a terminal, issue the following command:

```
apt-get -y install farpd honeyd-common libdbi0 libdumbnet1
libreadline5 librrd4 rrdtool ttf-dejavu
ttf-dejavu-extra iisemulator librrds-perl
```

Configuration

From a terminal, issue the following commands:

```
cd /etc/honeyd
mv honeyd.conf honeyd.conf.backup
vi honeyd.conf
```

Add in the following to *honeyd.conf*

```
create template
    set template personality "Microsoft
    Windows XP Professional SP1"
    set template uptime 1728650
    set template maxfds 35
add template tcp port 80 "sh /usr/share/honeyd/scripts/win32/
web.sh"
add template tcp port 22 "/usr/share/
honeyd/scripts/test.sh $ipsrc $dport"
add template tcp port 23 proxy $ipsrc:23
add template udp port 53 proxy 8.8.8.8:53
set template default tcp action reset
create default
```

```
set default default tcp action block
set default default udp action block
set default default icmp action block

create router

    set router personality "Cisco 1601R
router running IOS 12.1(5)"
    set router default tcp action reset
    add router tcp port 22 "/usr/share/
honeyd/scripts/test.sh"
    add router tcp port 23 "/usr/share/
honeyd/scripts/router-telnet.pl"

bind 192.168.2.100 template
    bind 192.168.2.200 router
    set 192.168.2.100 personality
    "Microsoft Windows NT 4.0 SP3"
    set 192.168.2.200 personality "IBM AIX
4.2"
```

Save the file and exit VI. By default Honeyd will run as the user 'nobody' with this configuration. So to make things happy we will need to change a few of debians defaults. In-order for the scripts to be able to log, the default permissions on '/var/log/honeypot' will need to be changed. As root issue the following commands:

```
chown -R nobody /var/log/honeypot
```

Running Honeyd

Running farpd

Farpd replies to any ARP request for an IP address matching the specified destination network with the hardware MAC address of the specified interface, but ONLY if the IP address is currently unallocated.

Any IP address that is claimed by farpd is eventually forgotten after a period of inactivity or after a hard timeout. The IP will be released immediately if a real machine claiming the IP address shows up on the network. When farpd is used in conjunction with Honeyd, it allows you to populate the unallocated address space with virtual honey pots. To start farpd listening on the network, in a terminal prompt. Issue the following command:

```
farpd -i eth0 '192.168.2.0/24'
```

As long as no errors come up within a few minutes, farpd will be responding to all unallocated IP addresses within the 192.168.2.0/24 network.

You should see something like:

```
arpd[1933]: listening on eth0: arp and
(dst net 192.168.2.0/24) and not ether src xx:xx:xx:xx:xx:xx
```

Running Honeyd

You can now run honeyd by issuing the following command:

```
honeyd -d -f honeyd.conf -p nmap.prints -x xprobe2.conf -a
nmap.assoc '192.168.2.100-192.168.2.200'
```

You should see something like the following, if everything is successful.

```
honeyd[2294]: listening promiscuously on eth0: (arp or ip proto
```

```
47 or (udp and src port 67 and dst port
68) or (ip and (dst net 192.168.2.100/30
or dst net 192.168.2.200/32))) and not
ether src xx:xx:xx:xx:xx
honeyd[2294]: Demoting process privileges
to uid 65534, gid 65534
```

Testing the honeyd

Now that Honeyd is running, your virtual honey pots should be alive and well. Best of all they are ready to be attacked! From another machine on your network, fire up a console and launch nmap at 192.168.2.100 & 192.168.2.200.

```
192.168.2.100
Starting Nmap 5.21 ( http://nmap.org )
at 2012-01-21 23:16 EST
Nmap scan report for 192.168.2.100
Host is up (0.038s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 0.42 seconds
192.168.2.200
Starting Nmap 5.21 ( http://nmap.org )
at 2012-01-21 23:17 EST
Nmap scan report for 192.168.2.200
Host is up (0.031s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet

Nmap done: 1 IP address (1 host up) scanned in 0.42 seconds
```

That's it you have now configured and setup two (2) low-interaction honey pots using honeyd. During this guide we have only covered a small portion of what you can do with honeyd. For more available configuration options check out the guide located at: <http://www.honeyd.org/configuration.php> or for more scripts check out some examples at <http://www.honeyd.org/contrib.php>, or you can also use these as a reference point for creating your own scripts.

Mwcollectd

Mwcollect is a honey pot for tracing and detecting malware and attacks. Mwcollect combines the best of honeytrap and nepenthes into one tool.

The authors of mwcollect have made a debian package available for those who do not wish to compile everything from scratch. At the time of this writing the latest available version can be found here: http://code.mwcollect.org/deb/mwcollectd-git_20111123-1_i386.deb

Installation

Dependencies

Libudns: http://ftp.us.debian.org/debian/pool/main/u/udns/libudns0_0.0.9-3_i386.deb

After downloading the above debian package, you can install and the other required dependencies from a terminal by issuing the following commands:

```
dpkg -i libudns0_0.0.9-3_i386.deb
apt-get install libnetfilter-queue1 libpq5
Installing mwcollectd
dpkg -i mwcollectd-git_20111123-1_i386.deb
```

Configuration

If everything goes well, you should now have a '/opt/mwcollectd' directory created with all of the related files installed. The main configuration file for mwcollect is located at:

/opt/mwcollectd/etc/mwcollectd/mwcollectd.conf

For the purpose of this demo, nothing in the defaults will need to be changed. See Default modules explained for information about the default loaded modules and their purpose.

IPTables NFQUEUE

Mwcollectd uses NFQUEUE to accept connections on arbitrary ports. One of the options below must be used in order for mwcollect to function properly.

If you wish for mwcollectd to use on specific IP address, use:

```
iptables -A INPUT -d $IP -p tcp --tcp-flags SYN,RST,ACK,FIN SYN
-j NFQUEUE
```

If you wish to run multiple NFQUEUE services on the same box, add in a unique queue number by using:

```
iptables -A INPUT -d $IP -p tcp --tcp-flags SYN,RST,ACK,FIN SYN
-j NFQUEUE --queue-num $QUEUE
```

NOTE: Make sure to edit /opt/mwcollectd/etc/mwcollectd/dynserv-nfqueue.conf accordingly.

```

Copyright 2009-2010 Georg Wicherski, Kaspersky Lab GbR <gw@mwcollect.org>
Copyright 2011 Georg Wicherski, McAfee GbR <gw@mwcollect.org>
This program is licensed under the GNU Lesser General Public License.

[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/embed-python.so with configuration /opt/mwcollectd/etc/mwcollectd/embed-python.conf...
[2012-01-22 01:11:44] INFO: Python 3.1.3+ with 2 extension module(s) ready.
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/dynserv-nfqueue.so with configuration /opt/mwcollectd/etc/mwcollectd/dynserv-nfqueue.conf...
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/dynserv-mirror.so with configuration /opt/mwcollectd/etc/mwcollectd/dynserv-mirror.conf...
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/filestore-streams.so with configuration /opt/mwcollectd/etc/mwcollectd/filestore-streams.conf...
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/filestore-binaries.so with configuration /opt/mwcollectd/etc/mwcollectd/filestore-binaries.conf...
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/shellcode-testing.so with no configuration...
[2012-01-22 01:11:44] INFO: Creating 1 shellcode testing threads.
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/download-tftp.so with no configuration...
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/download-curl.so with configuration /opt/mwcollectd/etc/mwcollectd/download-curl.conf...
[2012-01-22 01:11:44] INFO: download-curl with libcurl/7.24.0-DEV OpenSSL/0.9.8c zlib/1.2.3.4 libidn/1.15 ready.
[2012-01-22 01:11:44] SFAM: Loading module /opt/mwcollectd/lib/mwcollectd/log-file.so with no configuration...
```

Figure 3.

Default modules explained

- *embed-python.so* – This module embeds Python 3.x into mwcollectd.
- *dynserv-nfqueue.so* – This module enables the creation of dynamic servers using Linux netfilter queue or short NFQUEUE interface. If this module is enabled, traffic on unknown ports, regarding unknown vulnerabilities can be monitored.
- *dynserv-mirror.so* – Interacts with the attacker(s) in way to create network-dialogue in-order to trigger the proper downloading of shell code.
- *filestore-streams.so* – All network data that is seen by mwcollectd is stored per-connection stream recorders. Upon connection close, these are examined for shell code that might have been overlooked by the known vulnerabilities implementation or was sent during an unknown connection, such as in mirror mode. This module stores all of the traffic that has happened on these connections in the local file-system for further examination.
- *filestore-binaries.so* – Stores all malware samples in the local file system, uses MD5, SHA256 or SHA512. Default is SHA512.
- *shellcode-libemu.so* – libemu is a x86 emulation and shell code detection library. In mwcollectd, it is useful for finding out what a shell code does and how the attacking malware can be downloaded.
- *download-tftp.so* – This module solely listens to shellcode. download events and downloads malware binaries via the TFTP protocol.
- *download-curl.so* – This module listens to download.request and shellcode.download events and checks if these

reference http(s) or ftp URL's. If so, this module will then use the libcurl library to download these files and provide them as events.

- *log-file.so* – This module simply stores mwcollectd log output onto the local file system.

Extra modules explained

- *log-irc.so* – This module supports logging of output to an IRC server.
- *submit-mwserve.so* – mwserve is the malware aggregation service used by the mwcollect Alliance, it is a HTTPS based malware submission service that reports both new binaries and instances of already seen malware samples. The protocol mwserve uses is not complex, making it very easy to setup your own web server with a Python, PHP or Perl script as a backend to start collecting your own samples centrally.

Running mwcollectd

A good way to run mwcollectd is within a screen session so you can view the information coming in real time. You can do so by executing the following in a terminal prompt at the command line:

```
screen -S mwcollect
ulimit -c unlimited
/opt/mwcollectd/sbin/mwcollectd -l
```

If everything goes well, you should see something similar to the following.

```
[2012-01-22 03:44:54 EVENT] ["stream.request":10186cc1b4fcd5c0b00:000000000000000000] { port = "135", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] virtual void EmbedPythonModule::handleEvent(libnetwork::Event*) { 0x81e6c00, 0x805b190
[2012-01-22 03:44:54 EVENT] ["stream.request":10286cc1b4f085e0b00:000000000000000000] { port = "1723", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:1723.
[2012-01-22 03:44:54 EVENT] ["stream.request":0386cc1b4f7b5e0b00:000000000000000000] { port = "199", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:199.
[2012-01-22 03:44:54 EVENT] ["stream.request":10486cc1b4f15610b00:000000000000000000] { port = "3306", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:3306.
[2012-01-22 03:44:54 EVENT] ["stream.request":0586cc1b4fc0620b00:000000000000000000] { port = "5900", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:5900.
[2012-01-22 03:44:54 EVENT] ["stream.request":0686cc1b4fa9630b00:000000000000000000] { port = "139", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:139.
[2012-01-22 03:44:54 EVENT] ["stream.request":10786cc1b4f37650b00:000000000000000000] { port = "1720", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:1720.
[2012-01-22 03:44:54 EVENT] ["stream.request":0886cc1b4f16640b00:000000000000000000] { port = "143", address = "192.168.2.66", protocol =
[2012-01-22 03:44:54 SPAM] Spawning new mirror server on 192.168.2.66:143.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:23.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:25.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:254.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:113.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:995.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:443.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:111.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:8080.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:21.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:80.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:554.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:1025.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:3389.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:8888.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:445.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:993.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:22.
[2012-01-22 03:44:54 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:53.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:53.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:22.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:993.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:445.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:8888.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:3389.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:1025.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:554.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:80.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:21.
[2012-01-22 03:44:55 SPAM] Rate limiting SYN from 192.168.2.60 to 192.168.2.66:8080.
```

Figure 4.

Testing mwcollected

From another machine on the network, run a quick nmap scan against the host running mwcollected. On the screen session, you should see something like the following.

That's it you have now configured another Low-Interaction honey pot mwcollected. Now sit back and enjoy collecting some new malware for your collection!

Are honey pots legal?

There is three (3) main issues that are commonly discussed about the legalities of honeypots: entrapment, privacy, and liability.

Entrapment

Entrapment, by definition is "a law-enforcement officers or government agent's inducement of a person to commit a crime, by means of fraud or undue persuasion, in an attempt to later bring a criminal prosecution against that person." [Black's Law Dictionary, 7th Ed]

By this definition, entrapment is when you would trick or induce someone into doing something they would not normally do. Honeypots do not induce anyone, an attacker that discovers and exploits a honeypot are doing so on their own initiative. The attacker has usually already committed un-authorized activity in order to find the honeypot, the honeypot is merely just another system for them to attack.

Privacy

Privacy laws in the US may limit your right to capture data about an attacker, even when the attacker is breaking into your honeypot. The main challenge surrounding privacy laws and honeypots is the fact that there is no single statute that covers privacy. Instead we have things like: Federal Wiretap Act and the Electronic Communication Privacy Act. To make this issue more complicating, which legal statuses does one apply?

In the United States sometimes the state law can supplement federal law when privacy is concerned as it currently is in California. What happens if you have a honeypot in Georgia, but the attacker comes from California. Which laws should/will apply, Atlanta, California or the Federal? Without even bringing countries into the mix, you can see how convoluted this is becoming.

The use of honeypots affects the privacy issues as well. The reason for this is very important because of something

called the exemption under Service Provider Protection. What this exemption means is that security technologies can collect information on people and (would be attackers), as long as the technologies that are being used is serving a purpose of protecting or securing an environment. So in laymen terms, these technologies could possibly be exempt from privacy restrictions. As long as they protect or secure the environment in which they are used.

Federal Wiretap Act - <http://www.cybercrime.gov/usc2511.htm>
Electronic Communication Privacy Act - <http://www.cybercrime.gov/usc2701.htm>

Liability

The third issue is liability. Liability implies you could be sued if your honey pot is used to harm others. For example, if it is used to attack other systems or resources, the owners of those may sue. Liability is not a criminal issue, but civil. While this has been a discussion of legal experts for many years on the liability of an organization that has been compromised and in turn was then used to attack or compromise another system or organization. To date I have been unable to find any published decision addressing whether the operator of an insecure system can be liable to other entities for the misuse of the system by an attacker. So while liability will also be an issue surrounding honeypots, there is no recorded case of it happening with a compromised system.

My Opinion

First off I am not a lawyer, nor have I been one on tv. So my legal research has been provided via google searching.

I personally do not see any legal issues surrounding the use of a honeypot from a security or research purpose. Though everyone should exercise with caution and educate themselves on the local laws where it is intended to deploy the honeypot/nets.

JEREMIAH BROTT

Jeremiah currently holds a lead role with Access2Networks Toronto as an Information Security Consultant. In addition to holding numerous certifications, Jeremiah is also the professor of Malicious Code – Design & Defense along with Ethical Hacking at Sheridan Institute for the Applied Information Sciences System Security degree program.



Protect your important data before it is too late!



No-one likes to consider the worst case scenario, but are you prepared for a loss of all computer assets?

Computers are easily replaced, but your critical data and files are something money can't buy.

zebNet offers powerful, easy-to-use and leading backup solutions for all major web browsers and email clients which are designed to protect you as much as possible.

With a backup solution from zebNet you will always be protected from the worst case scenario at an affordable price, starting at just \$9.99

[Visit www.zebnet.us](http://www.zebnet.us) to be protected!

Highlighted features at a glance:

- Fast and reliable backup and recovery
- Self-restoring backup files
- Backup reserve copies
- Backup to any FTP server
- Scheduled backups on a regular basis
- Data migration between different computers
- Support for portable editions of your web browser/email client
- Create a portable edition of your web browser/email client
- And many more



Exclusive limited-time offer for you as a Hakin9 subscriber:

Get a **50% DISCOUNT** off any zebNet backup solution you wish by simply entering the discount code "**Hakin9**" in our store at **www.zebnet.us**

zebNet backup solutions are available for Microsoft Outlook, Windows Live Mail, Microsoft Internet Explorer, Mozilla Firefox, Mozilla Thunderbird, Mozilla Seamonkey, Google Chrome, Opera, Apple Safari, Postbox and Incredimail.

For any questions you may have, please get in touch with us directly at info@zebnet.us or visit www.zebnet.us



CYBER DEFENCE SUMMIT مؤتمر الأمن السيبراني

ENDORSED BY



APRIL 2ND - 3RD 2012

GRAND HYATT HOTEL, MUSCAT, OMAN

WWW.CYBERDEFENCESUMMIT.COM

DEFENDING YOUR VIRTUAL BORDERS

MIDDLE EAST IS GATHERING TO DEFEND IT'S CRITICAL INFRASTRUCTURE

TELECOM & IT SERIES

success is a choice
naseba



PLATINUM SPONSOR

Booz | Allen | Hamilton
strategy and technology consultants

GOLD SPONSORS



SILVER SPONSOR



BRONZE SPONSORS



MEDIA PARTNERS

For more information on being a part of this summit, contact: **Ali Khalid Rana**, Marketing Manager
Email: alir@cyberdefencesummit.com, Tel: +971 4455 7962

THE GAME OF GIVING MALWARE A NAME

MICHAEL BOELEN

Viruses, worms and trojan horses are a few common examples of malicious software. With a lot of new malware samples crafted and distributed each day, security researchers have a busy job discovering, analyzing and identifying them. So what happens when a wild malware sample is found and doesn't have a name tag yet? We'll have a look at this intriguing game of giving bad software the right name.

Malware in the wild

While running a honeypot, it suddenly gets "infected" with a piece of malware. After a quick look, it seems this catch could be an unknown species. After validating it with several malware scanning tools, it seems no one has ever seen it, or created an appropriate signature for it yet. So is this really a new piece of malicious software or simply one of the utilities used by the intruder? The hunt to find the identity of this odd visitor is about to start...

Analysis of malware, especially wild samples, can be a useful exercise. But what if you catch an unknown threat and want to get protected against it? In this article we have a look at the history and the challenges related to naming malware samples properly. We also dive into some basic analysis to find the right name for new catches, with the goal of finding our new friend's identity.

The rush

When anti-virus companies detect a new sample, they start a sophisticated process. This new threat is put on an operation table, to be dissected and analyzed. By means of static and dynamic analysis it will be checked to determine what the specimen is intended for. Such goals could be placing a backdoor, hiding and spreading itself to new victims, or simply becoming part of a botnet with the goal to assist in a denial-of-service attack at a later time. The goal of the malware sample isn't the only important thing. Also the form, its behaviors and the requirements to run, can be useful to the researchers. Last but not least, even if we have all this precious information collected, it would be good to find specific identifiers, to create a digital signature and provide proper detection in the future.

The analysis by the security professionals, usually working for anti-virus companies, consists of automatic scanning with intelligent software and where needed some manual research. The goal of the research is usually to get a clear picture on the

characteristics of the malware and find some sort of "identity" tag. The chance to find such a tag is usually very small though. This is the point where the creativity of researchers comes into play, with the goal of finding a proper name.

Researchers can give a sample every name they wish, as long as it follows some of the game rules of proper malware naming. One obvious, but important rule is that every specimen should get a unique name. While this sounds pretty simple, it's also the cause for a lot of confusion. Chances are very high that different anti-virus companies discover the specific sample at the same day. With the rush to analyze and name it, some malware samples will end up with different names.

The issue

Although one might think that multiple names for a single sample isn't such a big deal, this was certainly the case in the past. People had only limited and expensive access to the internet and therefore relied on finding information and details quickly. Users and system administrators wanted to find details about a particular virus or worm as soon as possible, to determine the cause of their instable or nonfunctional machines. With several aliases being used for the same specific piece of malware, people had difficulties in determining what exactly had infected their machines. Having the right name was, but also still is, crucial to find the antidote to an infection. We will have a look at how the anti-virus community dealt with this problem.

Working together

To deal with all those different malware names, several anti-virus companies formed a group called CARO (Computer Antivirus Researcher's Organization) [2], to research all kinds of malware. This group created a naming convention, to give future malware samples proper name tags.

The name would be crafted from several parts, shown in the scheme below. The scheme can be found on the website

of CARO and contains fields like the malware type, platform, family name, variant and if needed, some additional characteristics.

- `<family_name >`
- `<malware_type >://`
- `<platform >/`
- `.<group_name >`
- `.<infective_length >`
- `.<sub-variant >[[<devolution >]`
- `<modifiers >`
 - `!<locale_specifier >`
 - `#<packer >`
 - `@'''m'''<mailingmodifier.html |'mm'`
 - `!<vendor-specific_comment >`

When looking at scheme, the fields `family_name`, `malware_type`, `platform`, `sub-variant` and `vendor-specific comment` are the most commonly used. Let's have a small introduction for a few of these fields:

Family_name:

An unique name to identify which "family" a malware specimen belongs to. The family name is the easiest identification tag for users, to see what "thing" they are dealing with and giving them a clear reference name to search for. This part of the full name is usually what is being used on news sites, when a particular virus or worm is on the loose and getting new victims at a high rate.

Family names are only created when a specimen isn't directly based on earlier piece of malware. For other situations where the main code is the same, the scheme allows to add a variant name. It is common to have several malware samples within one family, especially when the author decided to tweak an earlier version to avoid detection.

Malware_types:

The type of the malware itself. Possible values are virus, trojan, dropper, intended, kit or garbage.

Surprisingly enough a worm is not a defined type. According to the specification it belongs to the "virus" type.

Platform:

The platform on which this particular sample will run, for example "W32", which is used as prefix for a Windows 32-bits binary, "OSX" for Macintosh malware and "Linux" for Linux malware. The platform "Unix" is used when dealing with malware which runs on several Unix alike platforms.

Although there is a clear convention on how to name malware, most of the anti-virus companies have created their own scheme based on the scheme of CARO. One of the reasons to use their own naming convention is that they already had a convention of their own. Another reason is the confusion about some of the more advanced malware samples. For example, a trojan horse containing a self-spreading worm part, but also with some DDoS functionality embedded. Each vendor has their own idea on the type, or at least the "main" type of the related malware piece, resulting in different names.

Beside the format differences these companies have, there are even more different values within the common fields like the platform. For example, Bitdefender uses "Win32" (instead of "W32"). This small change to the schema gives it a subtle, but important difference, when searching for a specific worm in

your favorite search engine. If we look at Microsoft, a relative newcomer in this field, it's obvious they didn't comply either. Their threat naming convention looks like this: *Type : Platform / Family Name . Variant ! Additional info.*

An example for a recent threat would be "Trojan:Win32/Sirefef.P". It's a trojan horse, running on a 32 bits Windows version and part of the Sirefef family, with the variant being "P" (implying it has some older siblings). In this particular naming sample, it becomes clear that anti-virus vendors not only have a different format, but often also completely different names. In the case of this "Sirefef" trojan, the aliases presented on the Malware Protection page of Microsoft shows the following aliases: *Win32/Sirefef.DK (ESET), Trojan.Win32.Jorik.ZAccess.qe (Kaspersky), FakeAlert-GA.gen.r (McAfee), Trojan.Gen.2 (Symantec), Cryp_FakeAV-56 (Trend Micro).*

With the exception of the ESET name, there is clearly a big difference in how the vendors identified this particular malware threat.

Another important aspect of naming differences has to do with 64 bits operating systems being more common now. Although a malware threat could be 32 bits, it still can fully ruin a 64 bits system. This is due to the 32 bits API's being available to the system, to support backwards compatibility. This could result in a malware specimen where there is one version affecting either platform, or several versions optimized for each platform. In the case of Sirefef, there is actually a 64 bits version identified by Microsoft (e.g. Trojan:Win64/Sirefef.B). Although one would expect the aliases to be similar to its 32 bits brother, the truth is otherwise.

These small examples give an impression on how messy this naming game can get. Although there is a set of guidelines available on how to name malware samples, it seems anti-virus vendors are not that strictly following it, or are confused as well.

Blasts from the past

To better understand how malware was named before, let's have a look at a few examples from the past.

Brain virus

About 26 years ago, two brothers from Pakistan named Basit and Amjad, created the first known virus for MS-DOS. These gentlemen were intrigued by the possibility of creating a fairly innocent "virus", with the intent to protect their medical software and target only those people making illegal copies of it. With the exception of wasting valuable memory space and slowing down floppy drives, the virus was safely programmed and instructed to infect media in floppy drives only. Since these side effects weren't always noticeable to those who copied the software, the "protection" didn't get much attention.

This virus had an interesting side. The names of the two brothers, the company name and even their telephone number were all included in the virus. People with an infected machine could use this information to contact the company to get a "vaccination".

The malicious code of the brothers was called the "Brain virus", not due to the smart body part used by the brothers, but the name of the company being listed within the virus as "BRAIN COMPUTER SERVICES". The name of the virus was at that time a logical chosen one. It clearly reflected a unique name, however with the risk that others would craft viruses to be named after them or their company as well.

Duqu

A more recent malware threat comes from Duqu. With a lot of reused code from Stuxnet together with a new stolen certificate, it had its share on the news sites in the last year.

Without going too much into details on this particular threat, the researchers from CrySyS in Budapest (Hungary), opted for the name Duqu after seeing file names created by the related key logger, starting with “~DQ...” [1].

If we look at some aliases on this threat from Trend software, we see different names for Duqu: *TROJ_SHADOW.AF [Trend]*, *TROJ_DUQU.ENC [Trend]*, *TROJ_DUQU.DEC [Trend]*, *RTKT_DUQU.A [Trend]*.

The first alias could indicate they renamed the threat, after Duqu becoming the primary name of this particular threat.

Sobig

When Sobig entered the digital world in 2003, it didn't take long before different anti-virus companies got their hands on a copy. This was mainly due to the high infection and spreading rate of the worm. Also in this case, we see that a company like Sophos originally named this worm “Palyh” (W32/Palyh-A), but changed it later to Sobig as well. The original “Palyh” was simply added as an alias.

Kaiten

Kaiten, or its alias Linux/Tsunami, is a tool to perform DDoS attacks. Pieces of its code were used in the Slapper worm. It infected many Linux systems which were running unpatched versions of the Apache webserver software.

In the case of Kaiten the name was assigned as a result of a line in the source code, mentioning this particular word.

```
Send(sock,"NOTICE %s :Kaiten wa goraku\n",sender);
```

The origin of the word Kaiten still remains a mystery. Some researchers think the tool could be of Japanese origin, but no clear proof has been found so far. The source of Kaiten was also the basis for a ported version to Mac OS X, with the name OSX/Tsunami.A (F-Secure). Although Kaiten is not focused on the Windows platform, Microsoft's anti-virus suite can detect the tool as well (detected as Backdoor:IRC/Kaiten.C).

Malware on the Unix platform

While the anti-virus companies agreed to some degree on how to name malware, several authors of Unix related tools didn't. The main reason is that these tools, usually created by open source developers, were not belonging to these multinational companies. With these individuals eager to discover, analyze and improve their tools to find malicious software, they didn't have company guidelines on how malware samples should be named. Another reason was the lack of anti-virus scanners for the Unix platform, with the existing software focusing only on the MS-DOS and Windows platforms.

With still a relatively low amount of malware being around for Unix based systems, individual samples can be identified easily and given a unique name. Usually it's only a small number of security researchers discovering a particular sample and checking the related paths, file names or strings and give the malware kit an appropriate name. For example in the case of rootkits, there was often a clear file name or string to be used as part of the name.

With more vendors building a Linux, BSD or Mac OS X scanner, malware on these platforms will receive more attention and is expected to be named according to the guidelines the world is accustomed to.

Naming rules

Now we had a look at all these differences and exceptions, we can still use the guidelines from CARO to properly name a malware sample.

As mentioned before, the most important part of the malware name, is the “family name”. When creating a new family name, make sure that it is easy recognizable for other researchers, or unfortunate end-users in the future.

A family name should not be given the name of a company, brand, trademark or the name of a person. Also the name of the author itself or the intended name by the author is usually not suitable to use. This is to avoid malware authors bragging about their “piece of art”. Beside names, do not use identifiers which restrict in time, location or origin either, as family names should be able to “contain” several variants. Using one of those specifics would force the need of creating several new family names, when a new version has a different time event, location or origin.

With these rules in mind and to exercise reasonable due care as a security researcher, you are advised to follow at least the following rules. Besides the proper naming tagging, it also decreases the chance for a malware author to achieve “fame” with his or her creation.

Practical steps

So if you are fairly sure you found a new malware specimen and are the first one to analyze and report on it, finding an appropriate name is the only thing remaining. Together with a good analysis of the malware, a proper name will make it much easier for other researchers to identify it more quickly.

Although the goal of this article isn't to dive deeply into malware analysis, some commands or resources are provided to help with the process of finding the right name. One particular useful Linux distribution to do this kind of analysis is REMnux [3]. It's time to proceed to the steps from catching a piece of malware, to finding the right name.

There is no use in submitting an alleged malware sample if it turns to be just a harmless script, document or binary. So first check the file type, size and possible readable text strings, to determine if any analysis is really needed. In general a file with a length of 0 bytes will not do much harm, nor does a normal file containing a password dictionary.

Related commands: *file*, *cat*, *hexdump*

Run several well-known and up-to-date virus scanners and let it test the related file(s). Another possibility is to use the website from VirusTotal [4]. This service enables users to upload a sample via the web page and check it against many anti-virus and malware scanning tools.

```
00000210 3e 3e 2f 43 6f 6c 6f 72 53 70 61 63 65 3c 3c 2f |>>/ColorSpace<</I
00000220 43 53 30 20 35 39 20 30 20 52 3e 3e 2f 46 6f 6e |CS0 59 0 R>>/Fonl
00000230 74 3c 3c 2f 54 54 30 20 36 31 20 30 20 52 3e 3e |t<</TI0 61 0 R>>
00000240 2f 50 72 6f 70 65 72 74 69 65 73 2f 45 78 74 47 |/Properties/ExtG
00000250 53 74 61 74 65 3e 3e 2f 54 79 70 65 2f 50 61 6f |State>>/Type/Pag
00000260 65 3e 3e 0d 0a 65 6e 64 6f 62 6a 0d 0a 36 30 20 |e>>..endobj..60 |
00000270 30 20 6f 62 6a 0d 0a 3c 3c 2f 4c 65 6e 67 74 68 |0 obj..<</Length|
00000280 20 35 33 39 2f 46 69 6c 74 65 72 2f 46 6c 61 74 | 539/Filter/Flat
00000290 65 44 65 63 6f 64 65 2f 54 79 70 65 2f 43 6f 6e |eDecode/Type/Conl
000002a0 74 65 6e 74 73 3e 3e 73 74 72 65 61 6d 0d 0a 65 |tenta>>stream..e|
000002b0 6e 64 73 74 72 65 61 6d 0d 0a 65 6e 64 6f 62 6a |ndstream..endobj|
000002c0 0d 0a 37 36 20 30 20 6f 62 6a 0d 0a 3c 3c 2f 53 |..76 0 obj..<</S|
000002d0 2f 4a 61 76 61 53 63 72 69 70 74 2f 4a 53 28 0d |/JavaScript/JS(|
000002e0 0a 65 3d 5c 28 5c 28 5c 28 27 76 32 31 27 5c 29 ||.e=\\(\\('v21'|\\
000002f0 5c 29 5c 29 2e 69 6e 64 65 78 4f 66 3b 0d 0a 63 |\\|\\).indexOf;..c|
00000300 3d 0d 0a 27 69 43 2c 37 78 6b 55 39 31 67 65 73 ||=..|c,7xkU9lges|
00000310 64 79 3a 72 77 6d 4d 7a 35 68 66 50 2b 71 3e 2a |dy:rwMzShfP+q*|
00000320 25 53 47 5b 76 6e 40 7d 75 61 5c 29 2d 74 26 4e |%SG(vn8)ua|)-t&N|
00000330 44 7c 70 7b 62 22 3d 6a 36 4b 6c 56 5c 28 45 3b |Dlp(b"*=6KlV(E;|
00000340 41 34 32 63 33 2f 6f 38 22 51 5d 57 3c 49 5f |A42c3/o8".QlW<L|
00000350 30 20 27 3b 0d 0a 6c 3d 27 6c 27 3b 0d 0a 65 3d |0 '.;..l='1';..e=|
00000360 65 5c 28 5c 29 5b 5c 28 5c 28 32 2b 33 5c 29 3f |e\\(\\|\\(\\(2+3\\)|
00000370 27 65 27 2b 27 76 27 3a 22 6a 6b 72 6c 32 33 6a |'e'+\\';';'kcl23j|
00000380 72 6b 6c 32 22 5c 29 2b 27 61 27 2b 6c 5d 3b 0d |rkl2"\\)+'+a+1;|..|
```

Figure 1: Check if the sample is really harmful

The terms-of-service state that all uploaded files may be stored and shared with other companies, including anti-virus vendors. For the security and protection of end-users it makes sense that this website collects possible harmful files and shares them within the security community. Be careful with uploading files containing personal or sensitive information.

If the file or files look suspicious, start with a static analysis of the file to determine its goals, origin or even references to other malicious code. In case of external references (e.g. an external URL), take precautions when downloading additional code or script. To protect your privacy and integrity of the system, it is preferable to perform such downloads in a sandboxed environment and behind an anonymizer service.

After the static analysis, it's time to do the dynamic analysis and check earlier assumptions. This includes investigating the possible actions a sample could perform. While debugging the code, check also for suspicious network traffic, disk IO or access to specific parts of the operating system (like memory, registry, and temporary directories). The real intent of the code should show up sooner or later.

The analysis can sometimes reveal special details found in unsuspected areas of the malware sample. Files could be storing some comment field, like binary files on the Windows platforms have additional fields as part of the PE binary file format. Also self-extracting archives (SFX) have this kind of meta-data sometimes available, to be extracted with tools like WinRAR or 7-Zip. Tar balls on platforms like Linux could give hint to a different original file name, which can be discovered with the 'file' command, like shown in the example below.

```
root@malware:/analysis/malware# file kit.tar
kit.tar: gzip compressed data, was "psybnc-linuxRO.tar", from FAT file-system (MS-DOS, OS/2, NT), last modified: Thu Sep 25 11:25:14 2008
```

The analysis phase of a malware specimen usually reveals particular strings or identifiers. When analyzing a wild sample, writing these specifics down can be useful, especially when having to think about a name later on. Related commands and software: strings command, debugging tools, disassemblers, packer identification tools (e.g. PeID).

If the particular sample still has no name, this is the time to give a name tag. Creating a new name could be as simple as combining a few syllables of common words, together with strings found in the malware file or kit. After creating a few fictitious names, look them up in your favorite search engine to double check that the name is not vulgar or otherwise insulting in a different language for example.

In general, names are easy to remember if they are pronounceable, short and consist of 2 or 3 syllables. Strings found in binaries or scripts could be used as input for final family name.

Remember to avoid the names which are intentionally added by the malware authors. Promoting their names will give them credit for their work, with the risk of them writing more samples and achieve "fame" under their peers.

After the analysis phase, it is time to properly document all findings. This helps in capturing the time and effort you invested in analyzing this particular malware sample. It gives others a better understanding of the intent and "tricks" used by this new threat. It also can be used to learn more about the steps taken in the analysis, possible missed areas or how to improve automating analysis in future.

Both the malware sample and your analysis, can be seen as a piece of art, with obvious the first one being more of a dark art. Still, both the malware and the analysis could be of great interest to other researchers. Be careful with distributing malware or malware snippets, especially if no proper detection is available yet.

After finishing the analysis of a malware sample, the community should be made aware of this new threat. By writing an analysis report and providing the malware sample to anti-virus vendors and other security researchers, it will give them quick insight in the related risk of the newly discovered threat. With all the technical details and a proper name, it shouldn't take them long to implement detection methods!

Conclusion

Although there isn't a real consent between anti-virus vendors on how to name malware, there are at least some ground rules. Most of them adhere to the naming scheme of CARO or use a similar format and fields. Regarding the naming confusion in the past, we can conclude that this issue is mostly gone, as most anti-virus vendors provide aliases of their colleagues. With the help of these aliases, it doesn't take long to find the "right" malware name. Also, it isn't the name that is the most important part, but the proper detection and providing some basic information for the threat. The web has usually enough pages covering the particular threat, including tips for proper removal. Specific tools to detect or remove new threats are usually quickly available as well, often provided by the well-known anti-virus vendors.

If you are ever in the opportunity to name a new malware threat, it's important to know the guidelines from CARO and the general tips provided in this article. In the end it's also the creativity of the analyst and the possible damage from this malware. Even the best name won't impress if the malware doesn't reach the news.

MICHAEL BOELEN

is working as a consultant for Snow B.V., a Dutch consultancy company with specialists in Unix and product groups focusing on system integration, networking, storage, database and security. As the original author of Rootkit Hunter, Michael gained extensive knowledge of malware threats on the Unix platform, including how to detect and protect against these threats. Currently he maintains Lynis, an open source auditing tool for Unix systems. In his spare time Michael likes to sport, visit friends, watch TV series or go for an occasional malware hunt. He can be reached via his website at <http://www.rootkit.nl>, the source for his software tools and related information, or via Twitter (@MBoelen).

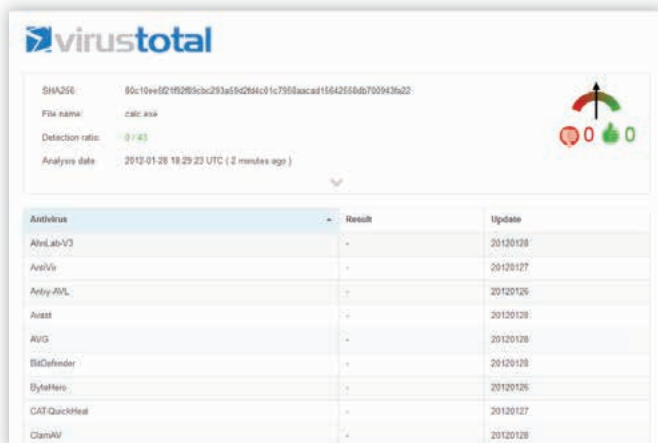


Figure 2: Check the sample against existing definition files

PROACTIVE NETWORK DEFENCE THROUGH SIMULATED NETWORKS

How to use some techniques and tools in order to deceive the potential intruders in our network

ROBERTO SAIA

Although most of the protections applied to our networks are based on an active approach that, more or less in real time, try to: contrast the attackers' attempts to violate the connected systems (for example, when we use a firewall) and find another way to operate, based on a diametrical opposite approach, a way that involves systems characterized by a passive attitude towards these events: we are speaking about simulated systems and networks, well known as Honeypots and Honeynets.

Albeit at first glance it may seem an irrational way to face this kind of problems, the passive approach is a part of a precise defensive scheme that, if well used, can realize a powerful tool for the network defence.

Despite in computer science literature we can find different solutions based on this kind of approach, the most known are the IDS (Intrusion Detection System) and the Honeypots (a composite word that sounds like 'a *honey pot* used as *decoy*'), in this article we will only focus on the latter type of solution, because it seems to offer more benefits in the proactive defence environment.

A honeypot-based solution realizes a credible simulation of a complete network environment where we can add and activate one or more virtual hosts (the honeypots) in various configuration: a network of honeypot systems is named honeynet.

The great potential of this defence mechanism arises when this operates within a real network with a certain number of real systems, because, first of all, it provides an effective distraction for potential aggressors and also, not least important, it allows us to get a lot of information about methods and techniques used during their attacks.

An honeynet is able to simulate thousands of different hosts and, depending on the software used, its simulation can be extremely sophisticated, involving a large number of parameters such as network services, delays, time of latency, and so on.

It is evident that there is a direct connection between the simulation level and the credibility of this one. For this reason, the choice of the software used must necessarily be followed by long and careful configuration activity.

Unlike what happens in the IDS systems that operate only with real systems and have many *false-positive* alerts, the whole traffic inside a honeynet can be considered as suspect, because it is not real, all inside systems are simulated and do not exist.

There is variety of honeynet/honeypot software, both commercial and open source, designed to run on different operating systems: in this article we will refer to only one, *Honeyd* for Linux, one of the first products of this type; a software with very interesting features which is distributed under the GNU (General Public license), therefore completely free (<http://www.honeyd.org>).

Although the development of this software has stopped a

few years ago and nowadays more sophisticated solutions are available, its simplicity makes it ideal to start taking the first steps in this area.

In this article we will see how to create a proactive defence scheme, something that works to distract the attention of potential attackers from real hosts and, equally important, to collect as much information as possible about the techniques and methods used by them during their unauthorized activity on the network.

Honeyd daemon

Honeyd runs on Linux as a daemon (so as a service) and permits to simulate a large number of virtual hosts, each of them with a specific IP address: these hosts can be configured by changing some files in order to perform certain services and in addition we can choose the platform service to emulate (Honeyd is able to emulate different operating systems).

Using a single system, Honeyd can simulate the activity of several distinct hosts (even thousands) and this allows us to implement a policy of masquerading of the real hosts, a policy oriented to confuse the potential aggressors: in a context with thousands of real and simulated systems that offer network services; against what system/service should an intruder launch the first attack?

Considering that the virtual systems respond to the activities of the attacker just like real ones, the probability that he/she is able to identify the real hosts with no other information is really negligible.

In addition to managing the simulated services on the same host, Honeyd can use a proxy mechanism in order to redirect the requests to another host where a real service of this type is running. However, the service will appear originated by the first host.

In a few words, Honeyd is an excellent open-source product be able to make a very efficient simulation: for example, we can freely modify the virtual network topology in order to create a very complex structures with a large number of router interconnections and even simulate the packet loss during communication by setting their latency. For security reason,

the host or the hosts used to run the Honeyd daemon should not be the production systems but hosts specifically dedicated to this particular activity.

We can install the package honeyd using the standard Linux modalities: after that we have downloaded the source from the official site of Honeyd (<http://www.honeyd.org>), we can install them by using an account with administrator privileges (login as *root* or by *sudo* command), the correct sequence is as follows:

```
tar -xzf <nome_del_pacchetto>.tar.gz
cd <nome_del_pacchetto>
./configure
make
make install
```

By this modality we need to solve manually unmet dependencies, otherwise we can install the package through the use of the command *apt* in order to automatically solve all needed dependencies:

```
apt-get install honeyd
```

When the installation is finished, we need to change our *IP-forwarding* setting in order to prevent that packets directed to simulated systems that are forwarded outside the host, creating duplications that would compromise the correct operation of the system. The command to execute is the following:

```
echo 0 > /proc/sys/net/ipv4/ip_forward
```

We can check the old value (or the new after the setting) using the following command:

```
cat /proc/sys/net/ipv4/ip_forward
```

The system is now ready to operate but before starting the operational phase, it is useful to take a look at the Table 1, where the Honeyd operating modality is summarized.

Table 1. Honeyd operating modality

Step	Description
1	The packets addressed to the simulated hosts are intercepted
2	Based on the configuration, each connection will bind to a specific simulated service
3	Before they are forwarded to outside, each packet is modified in accord with the operating system to be simulated

Honeyd is able to deceive scanning tools used to obtain systems fingerprinting, that is, those applications that are designed to detect which type of operating system is installed on a remote host.

To efficiently perform this task, Honeyd uses the same databases used by the scanning software, databases with a large collection of operating system signatures.

Honeyd creates virtual hosts operating within the range of addresses specified, it simulates the network stack of a real host and it is able to run services based on protocols as TCP, UDP and ICMP: by default all UDP ports are closed and, based on our configuration, Honeyd will generate an ICMP

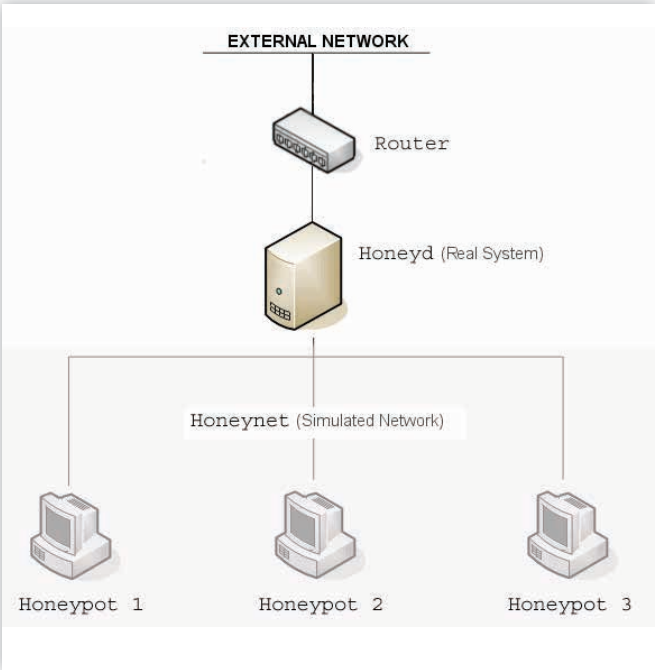


Figure 1. Functional diagram of the honeyd daemon

type 3 (Port Unreachable) message. The command line options we can use to run the Honeyd daemon can be viewed using the `'man honeyd'` command. The most important of them are summarized in Table 2.

Table 2. Some Honeyd running options

Option	Description
-d	Do not daemonize, and enable verbose debugging messages.
-l logfile	Log packets and connections to the logfile specified by logfile.
-p fingerprints	Read nmap style fingerprints.
-x xprobe	Read xprobe style fingerprints; this file determines how honeyd reacts to ICMP fingerprinting tools.
-a assoc	Read the file that associates nmap style fingerprints with xprobe style fingerprints.
-f file	Read the configuration in file.
-i interface	Listen on interface; it is possible to specify multiple interfaces.

The mentioned Nmap software (abbreviation of Network Mapper) is one of the most powerful tool currently available for network scanning, an open-source software freely distributed under the GNU-GPL (<http://nmap.org>).

Honeyd configuration

The Honeyd configuration file is based on the CFG (Content Free Grammar, http://en.wikipedia.org/wiki/Context-free_grammar) in according to the formalism BNF (Backus-Naur Form, http://it.wikipedia.org/wiki/Backus-Naur_Form), it uses a set of derivation rules written in the following format:

```
<symbol>:: = <expression>
```

In a few words, the symbol on the left of the rule can be replaced by any sequence among those present on the right (optional symbols are enclosed in square brackets).

For example, from reading this file it is simple to identify the different choices for the `'create'` directive:

```
creation= "create" template-name | "create" "default" |
          "dynamic" template-name
```

A simple example of this configuration file is the following:

```
### Simulation of a Web Server (Windows NT4)
create winsvr
set winsvr personality "Microsoft Windows Server 2003"
add winsvr1 tcp port 80 proxy localhost:80
add winsvr1 tcp port 139 open
add winsvr1 udp port 138 open
add winsvr1 udp port 137 open
add winsvr1 tcp port 135 open
set winsvr1 default tcp action reset
set winsvr1 default udp action reset
bind 10.0.1.51 winsvr1
### Simulation of a Cisco Router
create rtr1
set rtr1 personality "Cisco IOS 11.3 - 12.0(11)"
set rtr1 default tcp action reset
set rtr1 default udp action reset
set rtr1 uptime 2208866
bind 10.0.1.1 rtr1
```

The presence of obsolete operating systems such as Microsoft Windows NT, is a very powerful appeal to the potential

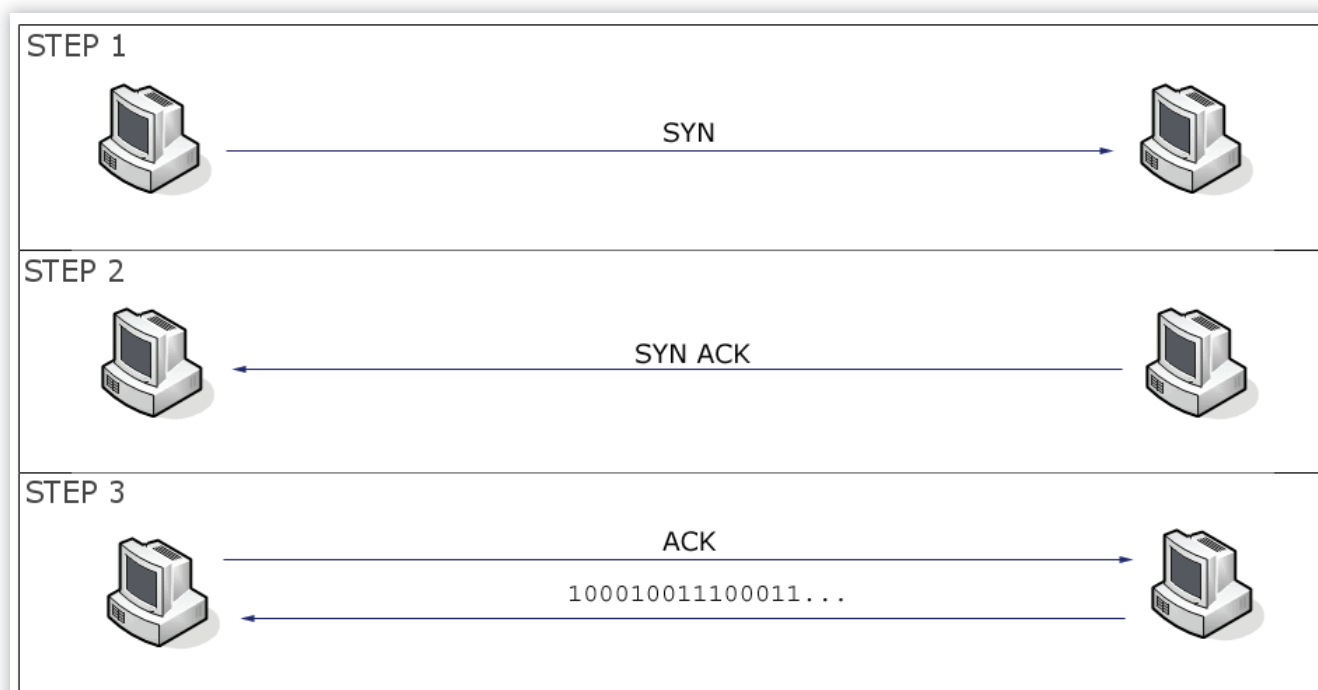


Figure 2. Three Way Handshaking

attackers. In the previous example, we have created two different personalities: the first one is a Web server based on the Microsoft Windows NT4 operating system that we have used in two virtual systems (honeypots) with IP addresses 10.0.1.51 and 10.0.1.52, the second one is a Cisco router device with IP address 10.0.1.1.

The *'create'* command defines a template, a model that will later be combined with one of the personality that we have just described in accord with a certain behaviour (ports and protocols used): in the previous example are active TCP and UDP ports 80, 139, 137 and 135.

Table 3 shows ports, protocols and related services that we have enabled in our simulation: we must be very careful in the choice of services to be activated in order to avoid inconsistencies (for example, activating a typical Linux service as *'finger'* on a Microsoft Windows host) and consequentially invalidate the simulation credibility.

Table 3. Emulate services

Protocol:Port	Description
TCP:80	World Wide Web HTTP
UDP:135	Remote Procedure Call
TCP:137 and UDP:137	NETBIOS Name Service
TCP:139	NETBIOS Session Service

During configuration, it should also be specified that the closed ports must respond by sending a flag type RST (reset) when the protocol is TCP and with a message *'Port Unreachable'* when the protocol is UDP.

As we can see, the emulation of a service (HTTP, Telnet, etc..) invokes a specific script written in Perl, for example, the HTTP service calls the script *'scripts/iis-0.95/iisemul8.pl'* in order to emulate a Microsoft web server IIS (Internet Information server).

Other directives as *'set'* and *'add'* are used to modify the behaviour of the simulated hosts (we have already shown how we can use *'set'* in order to define the default behaviour of each protocol). The possible replies of the TCP, UDP and ICMP protocols are summarized in Table 4.

Table 4. Possible replies of protocols

Azione	Descrizione
Open	Related to TCP and UDP protocols, it open all ports by default
Block	All packets are discarded by default without any notification to sender
Reset	All ports are closed by default: in case of TCP, it sends a RST response packet when receive a SYN; in case of UDP, it sends a 'port unreachable' ICMP message

In order to understand the contents of the table above, we have to brush up the concept of *'TCP/IP session'* between two hosts, a process named *'Three Way Handshaking'* (http://en.wikipedia.org/wiki/Transmission_Control_Protocol) that follows certain precise rules, rules summarized in Figure 2.

A simple example of the *'set'* directive is the following:

```
create myhost
set myhost default tcp action block
set myhost default udp action block
set myhost default icmp action block
```

It creates a new template named *'myhost'* which, by default, does not reply with all protocols: a typical use of these directives is the temporary exclusion of a number of hosts within a specific IP range, using the directive:

```
bind <ip_address> myhost
```

that we already seen before.

Using the *'delete'* directive we can modify *'on the fly'* an existing and running configuration in order to change or remove existing templates: when we make this, all active connections directed to simulated services will continue to operate regularly. Another directive is *'include'*, it is useful when we need to split the configuration file in more parts (in case of complex configurations).

Another important directive that we can use within the configuration file concerns the MAC address of simulated hosts.

Although honeyd can automatically do this, we can explicitly set a particular MAC address in this way:

```
set 10.0.0.1 ethernet "00-0F-EA-4C-07-93"
```

Otherwise, as we already said, Honeyd will set the MAC address in a smart way, based on the type of emulation (for example, in case of a Cisco router, will be selected a typical MAC address used in this kind of devices).

MAC Address

Acronym of *'Media Access Control'*, the MAC is an unique identifier assigned to network interfaces for communications on the physical network segment; MAC addresses are used in the *'Media Access Control'* protocol sub-layer of the ISO/OSI (International Organization for Standardization/ Open Systems Interconnection) Model (http://en.wikipedia.org/wiki/MAC_address).

Event logging

Honeyd can record network events in different modalities, it is able to generate very detailed records that contain all sorts of useful information related to connections in progress.

Honeyd can also send these information to a specific syslog server: this kind of server is used to collect all network logs in

```
Packet Level Logging
2008-05-05-19:26:40.8581 honeyd log started -----
2008-05-05-19:30:40.9109 tcp(6) S 127.0.0.1 60747 10.0.2.1 23 [SunOS 4.1 ]
2008-05-05-19:30:40.9112 tcp(6) S 127.0.0.1 60747 10.0.2.2 23 [SunOS 4.1 ]
2008-05-05-19:30:40.9113 tcp(6) - 127.0.0.1 60747 10.0.2.3 23: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9114 tcp(6) - 127.0.0.1 60747 10.0.2.4 23: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9114 tcp(6) - 127.0.0.1 60747 10.0.2.5 23: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9115 tcp(6) - 127.0.0.1 60747 10.0.2.1 113: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9116 tcp(6) - 127.0.0.1 60747 10.0.2.2 113: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9117 tcp(6) - 127.0.0.1 60747 10.0.2.3 113: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9118 tcp(6) - 127.0.0.1 60747 10.0.2.4 113: 44 S [SunOS 4.1 ]
2008-05-05-19:30:40.9118 tcp(6) - 127.0.0.1 60747 10.0.2.5 113: 44 S [SunOS 4.1 ]
...
Service Level Logging
2008-05-05-19:26:40.8582 honeyd log started -----
2008-05-05-19:33:50.4496 honeyd log stopped -----
2008-05-05-19:36:42.2082 honeyd log started -----
...
```

Figure 3. Example of collected data

a unique point (the syslog server), a standard service that usually operates through the UDP port 514.

There are two different levels of logging, PLL (Packet Logging Level) and SLL (Service Level Logging): respectively, they record the packet traffic and the service activities.

The next Figure 3 shows an example of information type collected in each modality.

Honeyd creation

On the basis of the above information, we now begin to define the configuration file that will allow us to activate a fully working honeynet.

The network topology we are going to create consists of 6 subnets connected by 4 routers, one of these subnets represents the entry point to the honeynet.

The first step is to determine the extension of the network and calculate its subnet mask: this operation is quite simple, we have only to convert to binary the first and the last address used, then perform a logical AND operation from left to right until the first zero, and from this point insert only zeros.

The correct subnet mask to use is the result value converted in decimal format (the Table 4 summarizes the characteristics of each subnet used).

Table 4. Used subnets

IP Address	Class	Description
10.0.0.0	C	external
10.0.1.0	C	Subnet 2
10.0.2.0	C	Subnet 1
10.0.3.0	C	Subnet 4
10.0.4.0	C	Subnet 3
10.0.5.0	C	Subnet 5

In Figure 4 we can see the result of the operation described above performed on the range from 10.0.0.0 to 10.0.5.255 (the first and the last address potentially assignable).

The resulted 255.255.248.0 is the subnet mask that we need to configure our system: the notation used by Honeyd is based on the number of bits set to 1 in the subnet mask: then we have to use '/ 8' for a class A network with 255.0.0.0 subnet mask, '/ 16' for a class B network with 255.255.0.0 subnet mask, '/ 24' for a class C network with 255.255.255.0 subnet mask and, in our case, '/ 21' to indicate a 255.255.248.0 subnet mask.

This information will be used for writing the configuration file that as first directive needs to know the network entry point: this is the border router with address 10.0.0.0/21. The next configurations regard the further subnets reachable from the outside: we want all subnets are reachable (in Figure 5 we can observe our honeynet topology).

The following instructions show the first part of the configuration file, the part where are placed all routing directives:

```
#border router interface and reachable networks
route entry 10.0.0.1 network 10.0.0.0/21

# Class of reachable addresses
route 10.0.0.1 link 10.0.0.0/24

# Networks and addresses directly reachable from
# border router 1
route 10.0.0.1 add net 10.0.1.0/24 10.0.1.1
```

```
route 10.0.0.1 add net 10.0.2.0/24 10.0.2.1
route 10.0.1.1 link 10.0.1.0/24
route 10.0.2.1 link 10.0.2.0/24

# Networks and addresses reachable from router 2
route 10.0.1.2 add net 10.0.4.0/24 10.0.4.1
route 10.0.4.1 link 10.0.4.0/24
route 10.0.0.1 add net 10.0.4.0/24 10.0.4.1

# Networks and addresses reachable from router 3
route 10.0.1.3 add net 10.0.3.0/24 10.0.3.2
route 10.0.3.2 link 10.0.3.0/24
route 10.0.0.1 add net 10.0.3.0/24 10.0.3.2

# Networks and addresses reachable from router 4
route 10.0.2.2 add net 10.0.3.0/24 10.0.3.1
route 10.0.3.1 link 10.0.3.0/24
route 10.0.0.1 add net 10.0.3.0/24 10.0.3.1

route 10.0.2.2 add net 10.0.5.0/24 10.0.5.1
route 10.0.5.1 link 10.0.5.0/24
route 10.0.0.1 add net 10.0.5.0/24 10.0.5.1
...
```

The first directive indicates the router interface connected to the external networks (10.0.0.1) and the reachable internal hosts (10.0.0.0/21). Subsequent groups of directives indicate networks and addresses reachable from each router and the last instructions that are the possible paths from the internal networks to the external networks.

The next step is to define some templates to be used for the simulated hosts (honeypots) that we decide to activate inside the subnets:

```
...
#Create default template
create default
set default default tcp action block
set default default udp action block
set default default icmp action block

#Server template
create winsvr
set winsvr personality "Microsoft Windows Server 2003"
add winsvr tcp port 80 proxy localhost:80
add winsvr tcp port 139 open
add winsvr udp port 138 open
add winsvr udp port 137 open
add winsvr tcp port 135 open
set winsvr default tcp action reset
set winsvr default udp action reset

#Client template
create winclnt
set winclnt personality "Microsoft Windows XP Professional SP1"
add winclnt tcp port 445 open
```

```
10.0.0.0      = 00001010.00000000.00000000.00000000
10.0.5.255   = 00001010.00000000.00000101.11111111
-----
255.255.248.0 = 11111111.11111111.11111000.00000000
```

Figure 4. Calculation of the subnet mask

```

add winclt tcp port 139 open
add winclt udp port 138 open
add winclt udp port 137 open
add winclt tcp port 135 open
set winclt default tcp action reset
set winclt default udp action reset
...

```

The first operation was the definition of a default template that will apply to all those hosts that do not use a specific template: inside our range of addresses, these kind of hosts will be inactive because our model does not provide any response to received requests (for all protocols).

The next step will be to define the router templates: we can choose the available models inside '*etc/honeypot/nmap.prints*' file: in our case we have chosen to simulate a 'Cisco 7206 router (IOS 11.1 (17))' and a 'DLink DI-604 ethernet router' (we must do this very carefully, writing the string of text exactly as in the file). The result is as follows:

```

...
#Routers template
create rtr1
set rtr1 personality "Cisco 7206 router (IOS 11.1(17))"
set rtr1 default tcp action reset

```

```

set rtr1 default udp action reset
add rtr1 tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set rtr1 uid 502 gid 502
set rtr1 uptime 2208866

```

```

create rtr2
set rtr2 personality "DLink DI-604 ethernet router"
set rtr2 default tcp action reset
set rtr2 default udp action reset
add rtr2 tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set rtr2 uid 502 gid 502
set rtr2 uptime 2309901

```

...

Now we need to use the '*bind*' directive in order to bind created templates to the hosts/routers used in our honeynet:

```

...
#Subnet 1 hosts
bind 10.0.2.200 winsvr
bind 10.0.2.101 winsvr
bind 10.0.2.102 winclt
bind 10.0.2.103 winclt

```

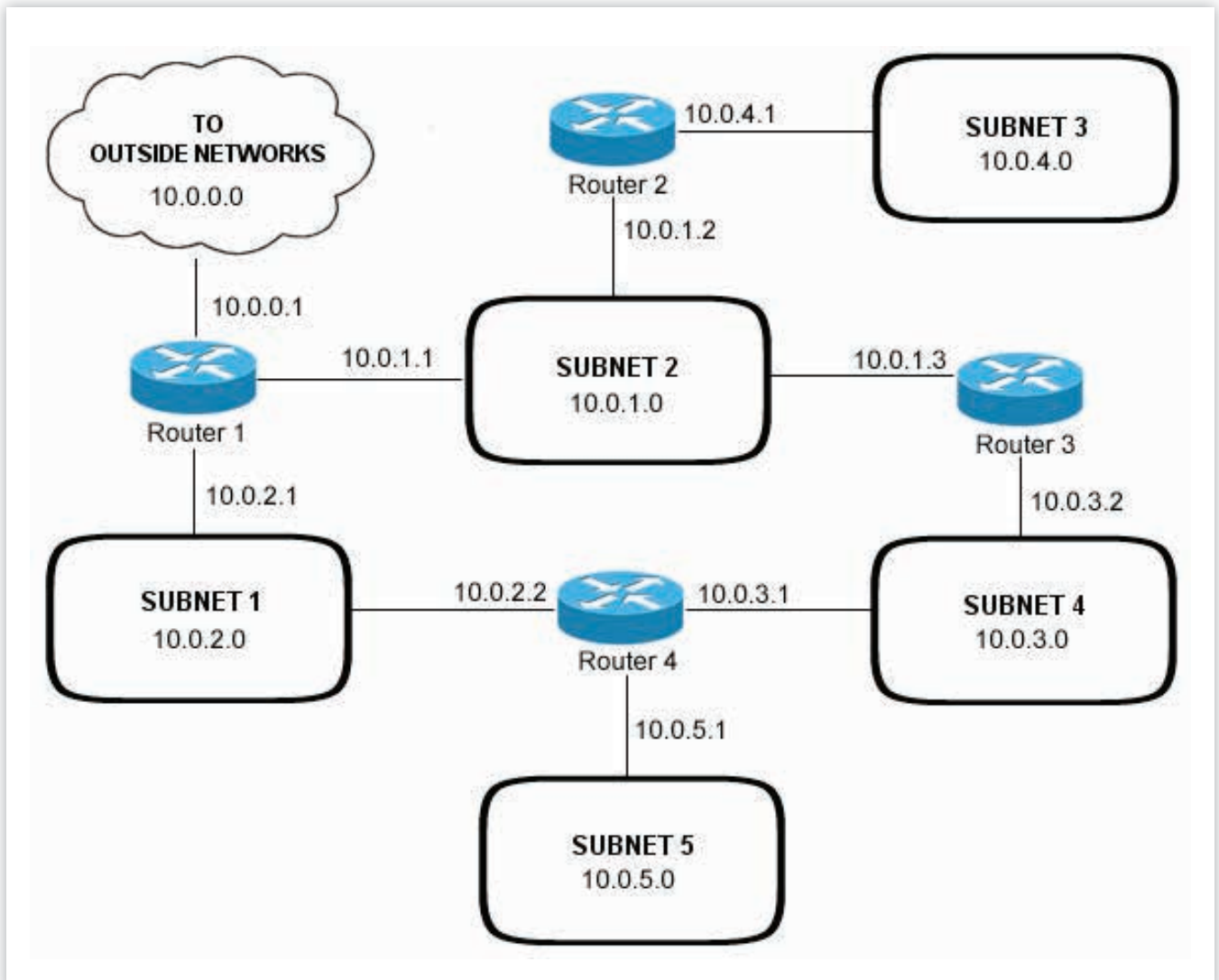


Figure 5. Honeynet topology

```
bind 10.0.2.104 winclt
bind 10.0.2.105 winclt
```

```
#Subnet 2 hosts
bind 10.0.1.100 winsvr
bind 10.0.1.101 winsvr
bind 10.0.1.102 winclt
bind 10.0.1.103 winclt
bind 10.0.1.104 winclt
bind 10.0.1.105 winclt
```

```
#Subnet 3 hosts
bind 10.0.4.101 winsvr
bind 10.0.4.102 winsvr
bind 10.0.4.103 winclt
bind 10.0.4.104 winclt
bind 10.0.4.105 winclt
bind 10.0.4.106 winclt
```

```
#Subnet 4 hosts
bind 10.0.3.111 winsvr
bind 10.0.3.112 winsvr
bind 10.0.3.113 winclt
bind 10.0.3.114 winclt
bind 10.0.3.115 winclt
bind 10.0.3.116 winclt
```

```
#Subnet 5 hosts
bind 10.0.5.111 winsvr
bind 10.0.5.112 winsvr
bind 10.0.5.113 winclt
bind 10.0.5.114 winclt
bind 10.0.5.115 winclt
bind 10.0.5.116 winclt
```

```
#Router 1 - Cisco 2600
bind 10.0.0.1 rtr1
bind 10.0.1.1 rtr1
bind 10.0.2.1 rtr1
```

```
#Router 2 - Cisco 2600
bind 10.0.1.2 rtr1
bind 10.0.4.1 rtr1
```

```
#Router 3 - DLink DI-604
bind 10.0.1.3 rtr2
bind 10.0.3.2 rtr2
```

```
#Router 4 - DLink DI-604
bind 10.0.2.2 rtr2
bind 10.0.3.1 rtr2
bind 10.0.5.1 rtr2
```

```
...
```

Honeyd simulation improvements

We can refine our simulated network by using particular directives. We have already used one of these during the router template configuration, where we had set the *'uptime'* parameter in order to emulate the device working time (this is a parameter used by the network scanners as Nmap to perform correctly their 'fingerprinting' operations):

```
set router uptime 2208866
```

As we can see in the following directives, we can also specify the bandwidth, the latency, and the packet loss over connections: a credible simulation of the behaviour of a real network:

```
route 10.1.0.1 add net 10.2.0.0/16 10.2.0.1
    latency 9ms bandwidth 10Mbps
route 10.2.0.1 add net 10.2.1.0/24 10.2.1.1 latency 6ms loss 0.5
```

Another useful directives are those relating to the *'dynamic templates'*. Based on certain parameters (summarized in Table 5) they allow us to dynamically change the behaviour of the simulated network.

Table 5. Dynamic templates parameters

Parameter	
Source Address	Connection source address determines the template to use
Operating System	Operating system of the source host, determined through a passive fingerprinting, determines the template to use
Time	Template is chosen by time criteria in order to simulate the switching on and off of the hosts within the network

A simple example of this directives is the following:

```
dynamic magichost
add magichost use linux if source ip=192.168.1.0/16
add magichost use windowsxp if source os=windows
add magichost use invisible if time between 10:00am - 12:00am
add magichost otherwise use default

bind source ip=192.168.1.0/16 10.10.0.200 cisco
bind source ip=10.22.83.0/16 10.10.0.200 juniper
```

where we have used what we have just said.

Nonlinear Assignment of addresses

In a real context the number of active honeypots should be appropriate to real hosts on the network and also their addresses must be assigned in a nonlinear way in order to confuse potential attackers.

Honeyd testing and logging

Although the configuration phase is completed, you must perform an extra step before you start the system: in the real system where the Honeyd daemon is running we need to configure the packets destination directed to the external network (10.0.0.0).

At this point we need to decide if we want to interface the system towards the outside, or use it only inside the host where it is running. For our tests we choose to use the last option that we implement in this way (all following commands must be executed as root):

```
route -n add -net 10.0.0.0/21 gw 127.0.0.1
```

After this, all traffic directed toward the mentioned network will be routed to the gateway 127.0.0.1 (loopback interface). Now we can run the daemon:

```
honeyd -f myfile.conf -d -i lo 10.0.0/21
```

The name of configuration file is 'myfile.conf'; by '-d' option we operate in debug modality (it operates in foreground in order to show information during its operation); by '-i' option we can specify what is the network interface and the class of addresses to be checked.

We can also logged all activities related to packet traffic and services activities, simply adding these options:

```
-l packets.log -s services.log
```

This creates two files containing this information. The correct operation of the system can be carried out through the standard operating system tools such as the 'ping' and 'traceroute' commands, respectively, to verify the reachability and the routing paths. However we can also use tools more sophisticated such as the Nmap port scanner.

For example, we can verify the correct routing of the packets (based on our configuration) in relation to the Microsoft Windows XP honeypot that belongs to the subnet 5 (10.0.5.0), the client with IP address 10.0.5.116:

```
user@system:~$ traceroute 10.0.5.116
```

We should obtain an output like the following:

```
traceroute to 10.0.5.116 (10.0.5.116), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.186 ms  0.160 ms  0.168 ms
 2  10.0.5.1 (10.0.5.1)  8.133 ms  8.117 ms  8.094 ms
```

As shown, the path is correct: from the border router 1 port 10.0.0.1 the packet is forwarded to router 4 port 10.0.2.2 and from this one it proceeds to the destination host 10.0.5.116 through the 10.0.5.1 port.

By the same way we can check all paths. The whole honeynet is shown in Figure 6.

A more detailed control is possible using the Nmap tool in order to verify, for example, the number of active hosts within a certain range of addresses:

```
user@system:~$ nmap 10.0.2.1-254
```

```
Starting Nmap 5.00 ( http://nmap.org ) at 2012-02-04 11:55 CET
```

```
Interesting ports on 10.0.2.2:
```

```
Not shown: 999 closed ports
```

```
PORT      STATE SERVICE
```

```
23/tcp    open  telnet
```

```
Interesting ports on 10.0.2.101:
```

```
Not shown: 997 closed ports
```

```
PORT      STATE SERVICE
```

```
80/tcp    open  http
```

```
135/tcp   open  msrpc
```

```
139/tcp   open  netbios-ssn
```

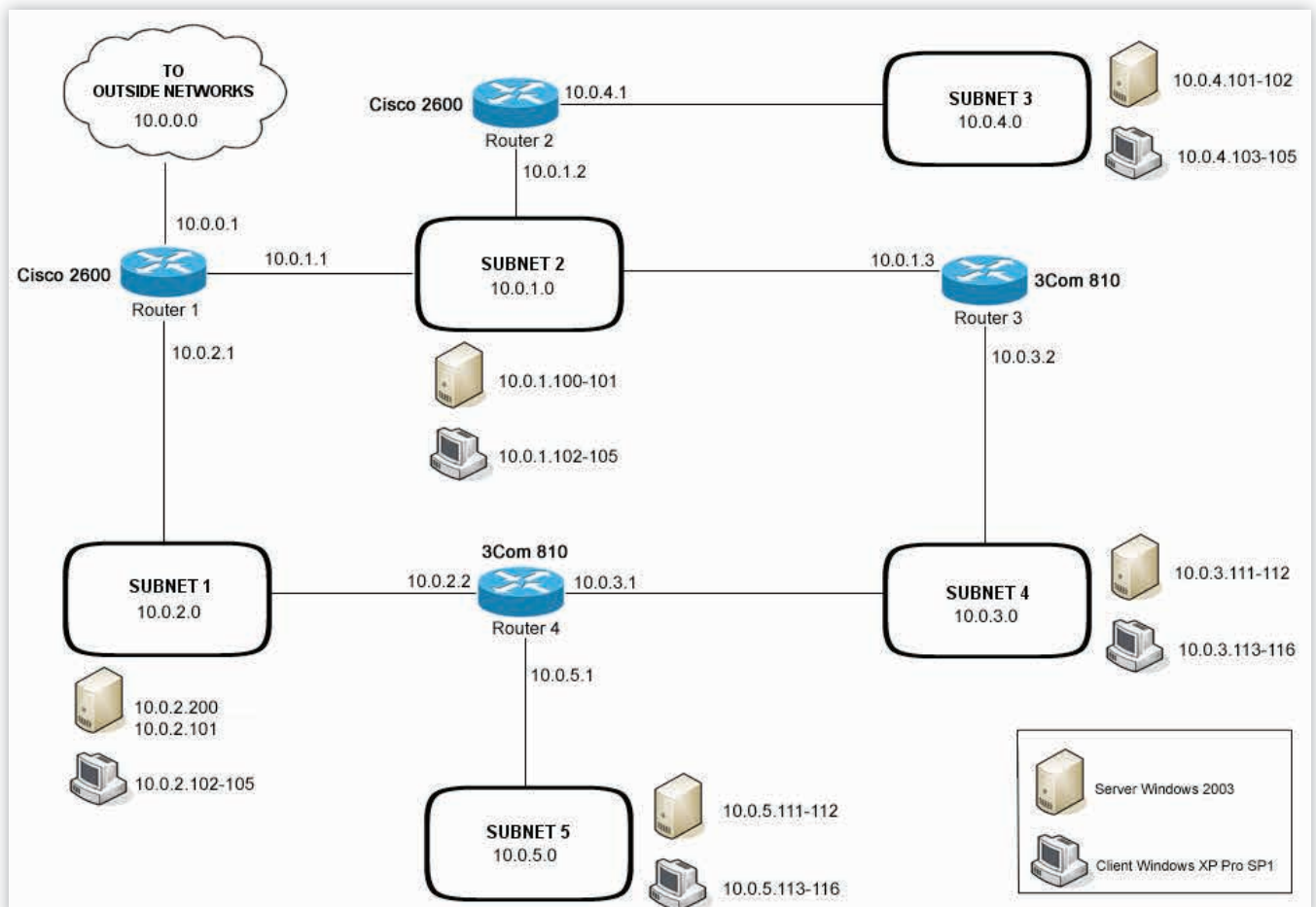


Figure 6. The whole honeynet.

```
Interesting ports on 10.0.2.102:
Not shown: 997 closed ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds

Interesting ports on 10.0.2.103:
Not shown: 997 closed ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds

Interesting ports on 10.0.2.104:
Not shown: 997 closed ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds

Interesting ports on 10.0.2.105:
Not shown: 997 closed ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds

Interesting ports on 10.0.2.200:
Not shown: 997 closed ports
PORT      STATE SERVICE
80/tcp     open  http
135/tcp    open  msrpc
139/tcp    open  netbios-ssn

Nmap done: 254 IP addresses (7 hosts up)
scanned in 10.54 seconds
```

We have already seen the use of the '-d' option in order to create two log files related to the packets traffic and services activity: we can have a simply access to content of these files using the 'tail' system command:

```
user@system:~$ tail -f packets.log services.log
==> packets.log <==
2012-02-04-12:52:06.8696 honeyd log started -----
==> services.log <==
2012-02-04-12:52:06.8698 honeyd log started -----
```

By this command we can get only the last rows of the files, therefore the latest logged events: their updates will be in real time and we can open two different terminal in order to obtain a better data view.

Conclusion

Based on what has been said in this article, we should be able to configure and run a honeynet in order to create an effective distraction aimed to confuse potential attackers of our network: related to our real networks, this kind of approach implements a proactive defence and presents two attractive advantages.

The first one is certainly the honeynet's ability to distract the attention of the attackers, making difficult the identification of real systems.

The second one, even more important, is the possibility to define the security policies (related to traditional defence systems as, for example, firewall and IDS) basing it on the Honeyd logs analysis, a precious source of information about the intruders behaviour. The logical process to follow is showed in Figure 7.

In conclusion, an honeynet offers us a different point of view on the attacker activities; we can log a large number of attacks and, much important aspect, each related activity can be logged step-by-step.

Based on this information, the operators can better understand the threats they face and how to defend against them.

Honeyd add-ons

The developer official website and several other internet places offer us numerous third-parties add-ons for Honeyd as, for example, scripts for emulation of a large number of services, as well as information and practical examples.

ROBERTO SAIA

Graduated in Computer Science, Roberto Saia professionally works in the ICT sector; for several years he has been managing computers network and security of a large national company; author of numerous books on programming, administration and system/network security, for some time his interest is mainly focused to the security environment, in the broadest sense of this term (<http://www.robertosaia.it>).

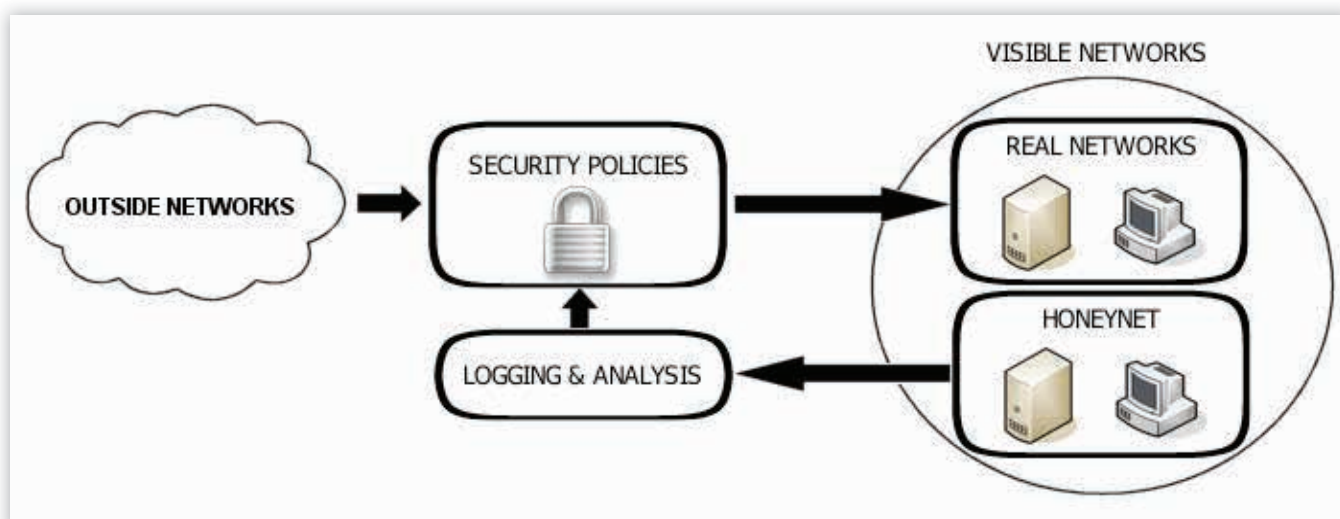
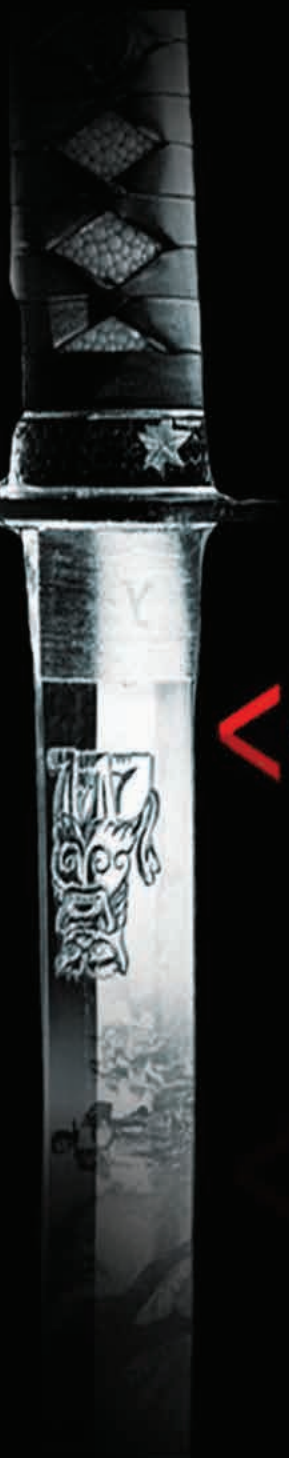


Figure 7. Proactive defence model

CODENAME: SAMURAI SKILLS COURSE



<< Penetration Test Training Samurai Skills >>

- You will learn Real World Hacking Techniques for Targeting , Attacking , Penetrating your target
- Real Live Targets (Websites , Networks , Servers) and some vmware images
- Course Instructors are Real Ethical Hackers With more than 7
- years Experience in Penetration Testing
- ONE Year Support in Forums and Tickets
- Every Month New Videos (Course Updated Regularly)
- Suitable Course Price for ONE Year Support
- Take Our course at your own pace (any time , any where)
- Our Course is Totally Different from Other Courses (new Techniques)

We have Real World Hacking/Penetration Testing Lab with Over 20 Real Target

USING HONEYPOTS TO STRENGTHEN NETWORK SECURITY

HARI KOSARAJU

Security Practitioners know that their Intrusion Detection Systems and host-based antivirus software is limited to already classified threats. They may be missing many other threats that are not detected. This article describes how they can better detect and characterize threats that may not have been caught by other means by using Honeypots.

Introduction

Honeypots have emerged as a new class of network security technology to address some of the shortfalls of existing solutions. In this article, we will first discuss the limitations of current network threat detection technologies. Next we will introduce various classes of Honeypots and how they differ. Third, we will examine how a potential attacker could detect a Honeypot and then, we will learn how Honeypots can be used to detect Zero Day attacks. We will conclude by discussing cloud based Honeypot architectures.

The Limitations of Current Network Security Devices

Network security is currently accomplished by firewalls, intrusion detection systems (IDS) and host based antivirus software. These systems are useful for blocking many known threats. Security practitioners know however that IDS and host-based antivirus are only as strong as their signatures. These signatures are crafted by vulnerability research teams at various security vendors. They may also be contributed by security professionals in the field. The problem occurs when a slight variant of a piece of malware for which a signature has been written starts propagating and this slight variation is able to thwart even a well crafted IDS rule. Another class of attacks are known as Zero Day Attacks. These attacks expose vulnerabilities in software packages that haven't even been publicly announced. Zero day attacks do not have signatures to detect them. The abundance of encrypted traffic now traversing networks exposes another limitation of current threat detection technologies. IDS cannot examine the payloads of encrypted traffic to detect threats. SSL Decryption technologies do exist but they work using a man in the middle approach and require client side certificate loading. Finally, IDS systems suffer from

false positives. This occurs when a signature is too generic and triggers alerts that are unfounded. This causes many security administrators to unload that particular signature and potentially leave their network somewhat unprotected. With these many limitations, the current technology leaves us vulnerable. It is clear that the black hats are not slowing down and "script kiddies" are more than happy to try out the latest threat SDK to attack our networks. How do we know when a new strain of malware has bypassed our IDS or when someone has exploited a zero day vulnerability and is not actively performing reconnaissance on our network?

An approach that can help in these cases is to deploy a Honeypot. A Honeypot is a resource which is meant to act as a system whose sole purpose is to be attacked. This means that any traffic that is entering it or probing it is by definition malicious and suspicious. The idea of Honeypots has been around for a while and was first described in the book "Inside the Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage" by Clifford Stoll. In the book, Stoll traces the actions of a hacker that has compromised the systems at Lawrence Berkley National Labs by setting up dummy machines and monitoring the hacker's actions. Honeypot technology has definitely progressed since this book was written in 1989 but the core principle remains the same : set up a dummy system that an attacker can't help but try and compromise. When trying to improve network security, it is important to not only be able to detect a new threat but also understand how an attacker will act once they have compromised the system. Another useful article by Bill Cheswick explains how he stealthily traced the actions of a hacker who thought he had exploited vulnerability in AT & T Bell Laboratories' Internet gateway. In order to be able to do this, it is very important for an attacker to not realize they are in a Honeypot. There is typical set of steps

that a hacker follows in compromising a system. The first step is a reconnaissance phase where network scanning tools are used to find vulnerable hosts running services that are known to contain exploitable interfaces. Then a compromise that has already been crafted is exploited on that system. This usually ends with the attacker getting some sort of a shell access to the exploited system. Once they get access to the system, they will typically cover their tracks by either erasing their logs, or installing a root kit of binaries that suppress logging. If you leave a system on the public internet, you will find that it is probed quite often. Further inspection will reveal that these probes are typically scans to find either open ports or to find specific services running. Often times these scans are automated but often also use specific tools like `nessus` or `nmap`. Hackers look for systems that have known exploits. Attackers may even use search engines to find vulnerabilities in specific web applications. This means, that there are a few ways to attract attackers to your system in order to learn more about them. Specifically, you may want to have specific ports open and even services running that have known exploits. This is just asking for trouble – but that is exactly what is wanted when driving attackers to a Honey pot.

Honey pot Variants

There are two major classes of Honey pots: Low Interaction and High Interaction. They differ in how many services they expose to attackers, whether these services are emulated or real and the amount of risk that comes in the event that a hacker is able to compromise the system. These classes of Honey pots also differ in how they are implemented.

Low Interaction Honey pots

Low Interaction Honey pots are Honey pots that emulate only certain services and thus limit the attack vector space. They are useful because they cannot be compromised to attack other hosts. In low interaction Honey pots, vulnerable services are emulated by using special handlers and not the real service. This tricks malware that happens to be probing a port that the service runs on into interacting with the service and having itself copied into a safe place where it cannot replicate or compromise the host. There are many examples of low interaction Honey pots including `Dionaea` and `Honey trap`.

How exactly do low interaction Honey pots actually trap a piece of malware? The first step of capturing the malware is to look for when a piece of malware tried to connect to a given TCP port. To understand this, we need to remember how the TCP Three way handshake works. The first step is for the TCP Server to bind to a specific port and start listening for connections. The client will send the server a TCP packet with the SYN bit set. This will then cause the server to send the client a SYN-ACK packet. Finally, the client will acknowledge this and send the server an ACK packet. Now what happens if a server is not in a listening state on that port and a client tries to connect? Well the server will send the client a TCP Reset (RST) which will tell the client to close its side of the connection. Low interaction Honey pots such as `Honey Trap` will sniff outbound packets for this TCP Reset. It will then intelligently start a listening service on that port. The next time the malicious caller attempts to connect, it will be successful. The other way `Honey Trap` does this is by having `iptables` send SYN packets to `Honey Trap` directly so that it can open the corresponding port. Now that we have a connection open to the attacker, there are a couple of things that can be done.

`Honey Trap` emulates service responses by sending the client back the contents of local response files for a particular service. It can also either mirror the traffic back to the client to have it basically attack itself. Another interesting option is to have it work as a proxy to send the malicious traffic to another dedicated machine.

If `Honey Trap` is running in Service emulation mode, all of the exploit data will be logged to the file system where it can be analyzed by a host of applications.

High Interaction Honey pots

High interaction Honey pots are typically complete systems running a full suite of services. They allow a high level of interaction with the attacker. None of the services being offered on these systems are emulated. The advantage of a high interaction Honey pot is that you learn more about an attacker's actions especially in the reconnaissance phase of their operations. High interaction Honey pots have their disadvantages as well. They are more difficult to instrument and they do carry much more risk in the event they are compromised. For instance, if the attacker is able to compromise this type of Honey pot and they launch an attack on another network from your Honey pot, you will be held liable. This risk is not present with low interaction Honey pots.

There are a few different options when considering construction of a high interaction Honey pot. You may choose to use a physical high interaction Honey pot which is basically a well characterized server. You also have the option of using a virtual high interaction Honey pot. There are a few common ways to do this: User-mode Linux and VMWare. User-Mode Linux allows you to run multiple virtual Linux instances as processes on a host computer. VMWare will allow you to construct a virtual machine that is running an operating system of your choice on a host computer. This provides flexibility with respect to the operating system of the high interaction Honey pot and also adds a layer of protection. What this means is that if an attacker is able to compromise the virtual machine, they are not attacking the native hardware platform. The other advantage of using a virtualized approach is that it is easier to revert to a "clean uncompromised" state. A final advantage of using a virtualized environment is that you can run Honey pots of multiple operating systems on one hardware platform. This enhances your attack surface and allows for the collection of information on more attack vectors.

Once we set up a high interaction Honey pot, we need to start gathering attack intelligence. There are a couple of ways to get information from a Honey pot on how the attack occurred. The first is to examine the log files. This can be problematic because the first thing an attacker does is to either modify system logs or delete them altogether. For this reason, it is important to first set up the Honey pot to use distributed logging. This ensures that the logs cannot be erased easily. The other piece of instrumentation that is useful is the use of a key logger. The command history that the attacker used in the exploit is useful. The other thing that is useful is an understanding of how they installed a rootkit. Typically, the login binary will be replaced with something that gives an attacker easy access to the system [8]. To learn which files that have been replaced by a rootkit, tools such as the `TripWire` file integrity checker are useful. You should use `tcpdump` to capture the packets that are entering and exiting the Honey pot. Note that this should not be lots of traffic because by definition your high interaction Honey pot should only be getting malicious traffic. Kernel

Figure 1: The pros and cons of High and Low Interaction Honeypots

Honeypot Type	Pros	Cons
Low Interaction	Less Risk of Host OS Compromise Lower Cost since services emulated and lighter weight Scales to many hundreds of instances well Can capture malware fairly easily	You need to be running vulnerable services which means running already known exploits Attack surface limited to the services you are running Limited understanding of hackers reconnaissance and other hiding techniques
High Interaction	Simulates real world conditions and helps learn about unknown attack vectors You can run multiple Operating systems on one physical host to increase coverage Can be as simple as setting another host on your network	High risk in the event of compromise Costly to implement on a large scale due to hardware costs and resource requirements Monitoring is more complicated since more services are typically deployed

level information is also useful. For instance, by instrumenting a layer between user space and all kernel level system calls, we can without a doubt track the attacker's actions. This is how the Sebek high interaction Honeypot works.

Honeynets

A Honeynet is a network of Honeypots that are characterized by a honeywall to divide parts of the Honeynet from other parts of the network. The honeywall is a firewall that prevents malicious traffic from leaving the Honeynet and attacking other networks. When talking about Honeynets, there is often a differentiation between Gen 1 and Gen 2 architectures. Gen 1 Honeynets have a simplistic firewall to block outbound malicious traffic. In Gen 2 Honeynets, this firewall setup is more sophisticated so that it can actually manipulate outbound traffic to make it benign. Honeynets capture three critical types of information about attacks which makes them very useful. First of all, there is a firewall log. In most cases, if iptables is used as the Honeywall, these messages can be logged to /var/log/messages. Another piece of the data capture component of Gen 2 Honeynets is a packet sniffer to record all traffic coming in and out of the Honeynet. Since this is malicious traffic, we do not expect large volumes of packet capture. The final component is a kernel module like Sebek to record the hacker's actions in a way that happens after traffic has been decrypted.

Detecting a Honeypot

It is important to understand how a Honeypot can be detected because savvy attackers will likely abort their mission if they believe they are being in a monitoring environment. Many malware now currently incorporate Honeypot detection within their logic so that they will "self destruct" on Honeypot detection. This makes it more difficult to contain malware so that effective signatures can be developed for it.

One of the most popular ways to log the actions of an attacker in a high interaction Honeypot is to use a tool called Sebek. Sebek is essentially a kernel module that can log keystrokes and operations to a log. Sebek actually has two components to it: A kernel module and a server piece. The server piece is designed to run on the honeywall whereas the kernel module is loaded on the Honeypot itself. Sebek is essentially a kernel level root kit. The reason it is implemented this way is because hackers had started installing their own binaries to circumvent user space logging. Since Sebek is implemented in Kernel space, this is not modifiable by the user. Specifically, it replaces the default read() function in the system call table with a new version and has the new version funnel data to a data logger function. This approach is interesting because it even deals with encrypted shell sessions because data is de-

crypt by the time it reaches the Sebek read function. Sebek actually does use a few methods to obfuscate itself from the attacker. First of all, it installs a second kernel module that acts to remove Sebek from the linked list of installed modules. Sebek also takes steps to hide the packets it sends from the Honeypot to the server. The way it does this is it by generating its own packets and not even using the raw socket interface upon which libpcap is based. There is still a problem though where if two Honeypots are installed on the same LAN, Honeypot A would be able to see Sebek packets from Honeypot B. Sebek circumvents this by installing its own Raw Socket implementation that silently ignores all Sebek packets. It is now clear that many of the obfuscation techniques that have been coded into Sebek have been broken. For instance, it is possible to detect kernel modules even if they have been cleaned. This approach has to do with searching for the kernel module header structure which happens to still be in memory. Another way to detect Sebek is to look at the System Call Table on a host and to compare that to a normal configuration. This approach looks for modified function pointers. Many security researchers know are the limitations of Sebek as a high interaction Honeypot and have been working on alternatives such as Qebek. That being said, if the attacker is not savvy enough to look for evidence of the Honeypot using these principles, it may still stay hidden. There are even techniques to disable Sebek on the windows platform, which make it even less desirable.

It is also possible to detect that a Honeypot is running in a virtualized environment. The key thing to remember about running in a virtualized environment is that execution timing is altered because of the virtualization layer. For example, if you look at ICMP Echo response times between a virtual machine and a physical machine, there is a delay. This is likely due to the fact that the packet traverses the TCP/IP stack of the virtual machine and not just the physical host machine. More instructions must be executed which therefore adds delay.

Another way that attackers can determine if they are in a virtualized Honeypot environment is by using Execution Path Analysis. EPA is enabled by connecting the syscall handler (int 80) and the debug exception handler (int 1) in the IDT (Interrupt Description Table). Then, by setting the TF bit (mask 0x100) in the EFLAGS register, the new handlers are able to count each SIGTRAP generated when an instruction is executed. This approach does have a few limitations in that the attacker needs high level privileges to execute these changes and also, the modification of the system calls are not covert.

Virtual machine detection can be done in other ways that namely examine, file and registry artifacts, running processes and directories. It is also possible to examine memory to find evidence of a virtual machine. For example, in host machines,

the interrupt descriptor table is in low memory while on the guest machine, it will be usually be higher in memory. This approach can be extended to look at the location of the GDT (Global Descriptor Table) and the LDT (Local Descriptor Table) in addition to the IDT (Interrupt Descriptor Table). The third way to detect a virtualized environment is to look for virtual hardware. In a Linux environment, one standard check is to look for VMware specific naming in the proc filesystem and also to look for well known VMWare devices. Another interesting way to detect that you are running within a virtual machine is to look for Virtual Machine specific instruction support. Virtual machines support instructions that are not available on a host machine. These instructions are namely there to facilitate guest to host interaction. There are tools to discover this trait by attempting to execute the VM specific instructions and then seeing if their exception handler was triggered. If it was, then that means the instruction was not supported and we are executing on a host. If the exception handler is not triggered, it means that we are running in a VM.

The point with discussing how an attacker may detect a virtual machine is to shed some light on the sophistication of current malware. Many of the latest types of malware are able to exploit these detection schemes and not execute their most sensitive operations to prevent detection. These methods can and have been tightened up in many cases but are the evidence of a Honeypot that a hacker will look for after gaining access to a system. It really is a constant back and forth to hide virtual machines from the latest hacker detection technique. You should assume that the hacker knows everything that is written here because it is all publicly available informa-

tion. Honeypots are no panacea for catching the bad guy but that does not mean they are not valuable.

Detecting Zero Day Attacks using Honeypots

One of the limitations of most cyber security technology is detecting the zero day threat. These threats have yet to be characterized and thus are not part of the rules of any security vendor's threat database. Honeypots can help in detecting these threats. This section of the article describes the latest work with respect to detecting zero day attacks. Some of the latest malware is self replicating and mutating. This means that it does not fit into readily defined signatures. Honeypots can help us characterize and protect against this threat.

Some interesting work is being done to determine how to detect zero day threats by using Virtual Honeypots. The Argos Emulator is a virtual high interaction Honeypot designed to capture Zero Day Attacks. Argos employs QEMU which is an open source emulator and virtualizer and uses an idea called Dynamic Taint Analysis to determine when network traffic is executed. If you think of what a zero day attack is, it is basically when network traffic payload ends up being executed on the host. A processor's conventional control flow is diverted by the attacker and code that they have injected is executed, which often launches a shell for the attacker to login and further compromise the system by disabling logging or installing a rootkit. Argos tries to detect these instances. By correlating the network traffic with information logged by the QEMU emulator, Argos is also able to generate intrusion detection signatures to prevent the attack which are immune to changes in the payload. This means they cannot be manipulated to

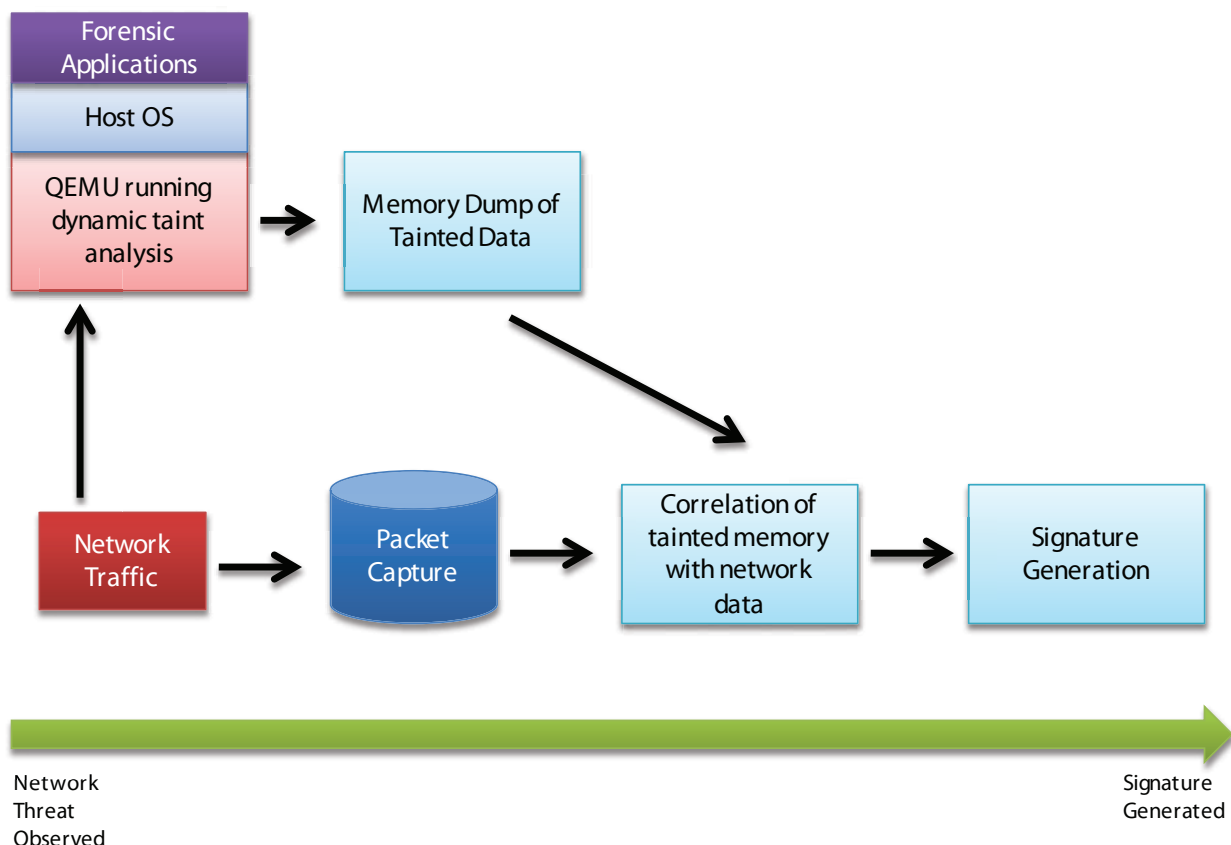


Figure 2: How Argos Detects Zero Day Attacks and Generates Signatures

produce an undetected variant. This approach apparently has very low false positives.

Latest Trends in Honeypot Architectures

Antivirus companies have been using Honeypot and HoneyNet technology to capture and produce signatures for malware for quite some time. For instance, Avira has deployed a distributed HoneyNet to capture and analyze malware samples. The problem they had faced was that malware tends to try and infect hosts with similar IP addresses because there is a higher probability that the IP will have been allocated. The Avira HoneyNet was a low interaction Honeypot which emulated various vulnerabilities to gather worm variants that used that particular exploit. These samples are transported from various clients to a centralized server for further analysis. This is an example of a distributed HoneyNet. Taking this idea a step further is the use of cloud computing concepts for Honeynets. This is interesting because it couples the virtual machine based aspects of virtual Honeypots with this distributed data collection architecture described by the work done at Avira and others. Migrating Honeynets to the cloud is the future as global companies continue their migration to cloud based application architectures. This also means that cloud based security is becoming more critical. For instance, many cloud customers would like to ensure that their virtual instances have not been compromised. Some researchers have proposed what a cloud based Honeypot architecture would be and have proposed Honeypots as a service. It is not public at this point whether Amazon EC2 or RackSpace have implemented a Honeypot as a service since it is not currently listed on their sites. Amazon web services do detect when port scanning is happening and it is explicitly against their acceptable use policy, however it is not clear what tools they use to detect it. Further, it is apparently not possible for one tenant of the Amazon Web Services to sniff the traffic of another tenant by putting their Virtual Machine in promiscuous mode because the hypervisor will not deliver frames to them. Even two virtual instances owned by the same customer cannot listen to one another's traffic.

Conclusion

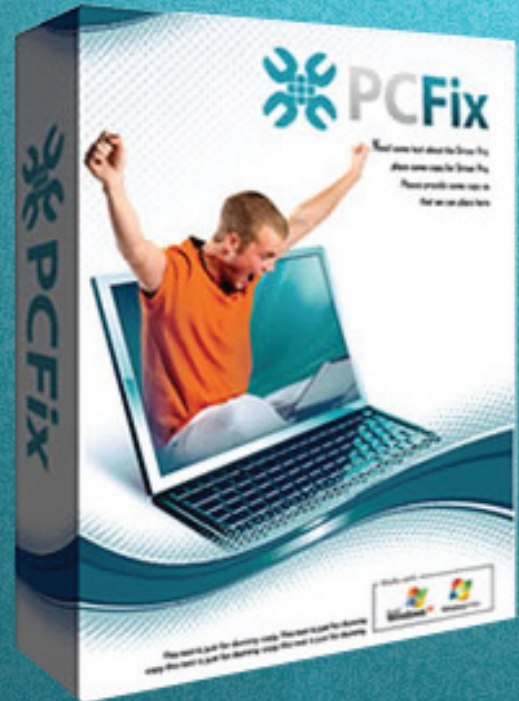
In summary, Honeypots come in many different varieties which each have their own pros and cons. They are very valuable tools for finding network attacks that otherwise go undetected with conventional network security tools. Distributed Honeynets, cloud based Honeypots and Honeypots running in emulated environments to detect zero day attacks are interesting areas that should be monitored closely for their future contributions to improving network security.

HARI KOSARAJU

has over 10 years of software engineering experience focused on Linux Application Development in C/C++, Real Time Embedded Systems, Protocol Design and Deep Packet Inspection. He is currently a lead engineer on the Mantaro SessionVista™ product line of Network Intelligence Technology. This product line is used for Network Forensics, Lawful Intercept and Cyber Security through industry leading deep packet inspection at high data rates. Hari's interests lie in understanding network based threats and developing the technologies to detect and stop them.

Hari holds a bachelors degree in Systems and Computer Engineering from Carleton University in Ottawa, Canada. He also completed a joint MBA/MS at the Robert H. Smith School of Business at the University of Maryland, College Park.

PC Fix



Fix Windows Registry & Repair PC Errors!



Before you continue:

- ✓ Free scan your Computer now!
- ✓ Improve PC Stability and performances
- ✓ Clean you registry from Windows errors

Instant Scan

You've been using us for years.



World-Leading Security Consulting.



Email sales@mdsec.co.uk

Tweet [@MDSecLabs](https://twitter.com/MDSecLabs)

www.mdsec.co.uk

CLIENT HONEYPOTS

ING. MICHAL SRNEC, ING. MIROSLAV LUDVIK, PH.D.

Development of security tools has been on the rise in recent years. The main reason of that is the wide variety of attack trends against computer systems. This new technologies like intrusion detection systems (IDS), antiviruses and firewall, help to address this issues. One of this new technologies is honeypot.

Honeypot is pretty new technology which use different technique to help address security problems. One of the many definition of honeypot is “a resource whose value is in being probed, attacked or compromised”. Another resource define honeypots like: “A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource”. In other (user friendly) words: Underlying strategy is simple but really powerful - to allure potential attackers to fake network node and tracking the attacker operations. Based on this observation, system administrators can build the security policy.

The aforementioned definition fits very well with traditional honeypot – server honeypot. This traditional honeypot waits for attack from remote side. Of course, it not only waits, but tries to allure potential attacker using different techniques like open and unsecured ports, emulate another OS etc. As for the types of attacks, we must consider that most malicious attacks occur from client-side attacks. Client-side attacks refer to attack against end-user. Given this, it has been necessary to develop new kind of active honeypot, which can track and analyze client-side attacks. This new type of honeypot is called client honeypot. This active honeypot emulate (manually or automatically) the normal series of a regular user steps. Client honeypots aim to identify malicious websites which want to abuse client application vulnerabilities.

This ultimate concept of client honeypots was firstly articulated by Lance Spitzner in 2004. Nowadays, we know several implementation of this new kind of honeypots like Honeyclient, HoneyMonkey, HoneyC and Capture. However, the progress in development of the client honeypot is conspicuous, this technology is still toddling.

Client honeypots vs. server honeypots

The main difference can be formulated like: rather then “passively” wait for attacks, client honeypot actively searches for malicious websites. Client-side honeypots should by designed according to server protocol (the client depends on server it is working with). This idea can by achieved by simulating a

human behaviour to determine whether the server is exploiting the client system. This is one of the reason why they must be active, which means send requests to server, and analyze the response by using various techniques. Main difference between this two kinds of honeypots can be summarized as follow:

- Type of simulated software: client-side honeypots simulate client software and not the server based services to be attacked
- Activities: actively initiate interaction with remote servers to be attacked
- Identifying: by server honeypots all traffic can by considering as malicious. Client honeypots must discern which server is malicious and which is not.

Client honeypots approaches

As we proposed the main goal of client honeypots is to detect and identify any malicious activity coming from the Internet. Ideal client honeypots should be summarized as follows:

- Detect any know and unknown threats against any client user application and should be able to check various URLs, with rate zero false positive.
- Should detect the attacks in real-time.
- Should be able to change detection and secure policy rules as a response to current situation

General approach involve two phases:

- Crawling: this phase is common for all client honeypots; firstly we must look for malicious websites.
- Detecting: in this phase honeypot identifies if the queued sites are malicious or not. To do this, client honeypots use two methods:
 - Pattern-Matching: used by low-interaction client honeypots
 - State Change Check: used by high-interaction client honeypots

Crawling

At first, honeypot needs to actively search through the network to find servers to interact with. Given the size of the Internet (which involve millions servers) is not a trivial issue. Usage of this techniques needs to satisfy three requirements:

- Obtaining high speed
- Avoiding overloading
- Avoiding sample bias

Crawling speed depends on more factors – hardware, bandwidth and crawling algorithm.

The second issue is information overloading. Crawling algorithm should avoid crawling of the same content. Avoidance of crawling of the same content is important because modern websites are machine-generated with URL as a transport layer for information. Some client honeypots (aka MonkeySpider) uses URL normalization techniques to avoid unnecessary overhead, while downloading the same content multiple times.

Third issues is sample bias. Various techniques/schemes can be used to determine and sort website for inspection. Some queuing techniques depend on premise that malicious websites contain particular type of content such as pornography. Another algorithms create queues of web pages based on keyword or extracted links from SPAM. However, this scheme will lead to miss many malicious websites, because attacker can use less suspicious keywords to build malicious website. In generality, this approach can not cover large scope of the Internet, because most malicious web pages do not fit to this schema. The second method uses hyperlink structure on retrieved documents to access other websites. However, this method has its vices: popular websites are linked with higher rate and random crawling would lead to bias in the sample. Various researchers try to solve this shortcoming by adjustments based on page popularity. Although the bias is reduced, it is not removed. Further, this

method misses any pages for which no hyperlinks exist. Another method can be used to determine scope of inspected website. Third method lies on generation random IP address and check the presence of web servers. If server is present, crawling algorithm selects random page from hosted web pages on the web server. From the methods mentioned, the last one (generating random IP address) will create most unbiased random sample.

Pattern-Matching

Client honeypots use simulated clients and they do not use fully functional operating system – they often emulate web browsers, or web crawlers. They have only limited abilities for interaction with attackers. Low-interaction client honeypots send HTTP requests to the web servers and apply “signature-based or heuristic methods” to the servers response to detect malicious server. Applying this method can directly detect security violation. A honeypot which uses this method can be expected to quickly detect attacks. However, some unimplemented attacks will be likely missed by using this method. Examples of client low-interaction honeypots which use pattern matching method are Spybye, HoneyC, MonkeySpider.

Detecting – State changes Check (Integrity check)

Client honeypots which use this approach are based on high-interaction honeypots – attacker interact with real system rather than with simulation. This method (integrity check) is the only way that allows high-interaction honeypots detect security violations. This method relies on access to suspicious websites and simultaneously detects whether any changes happen on the client system. If any changes are given, the system has been exploited. Detection, whether the system has been exploited involves monitoring the following:

- Filesystem activities.
- Registry entries.
- Processes.
- Network connections.
- Memory. This is the ultimate state change check.

However, such checking is not easy to implement.

Current high-interaction client honeypots are limited to monitor filesystem, registry and processes in order to obtain faster and easier implementation.

With this method we have to pay more attention to avoid false positives i.e. many websites create cookies on our system to save information about our visit. Thus, we should have some exclude list to avoid false positives.

Figure 1 presents an overview of high-interaction honeypot. User can set various parameters using front-end configuration. These parameters can, for example, be keywords, depth and breadth of crawling or number of URLs after which client honeypot stops its execution. Browser simulator module emulates the web browser, URL analyzer handles the dialog boxes and integrity check module checks the state of our system to detect any changes. All log files are stored in remote database to enable centralized logging. Client honeypot can run in virtual machine like VMware, which is helpful when we need reset the machine to clean state.

Detection issues

Various detection problems can occur when we use detections approaches discussed before.

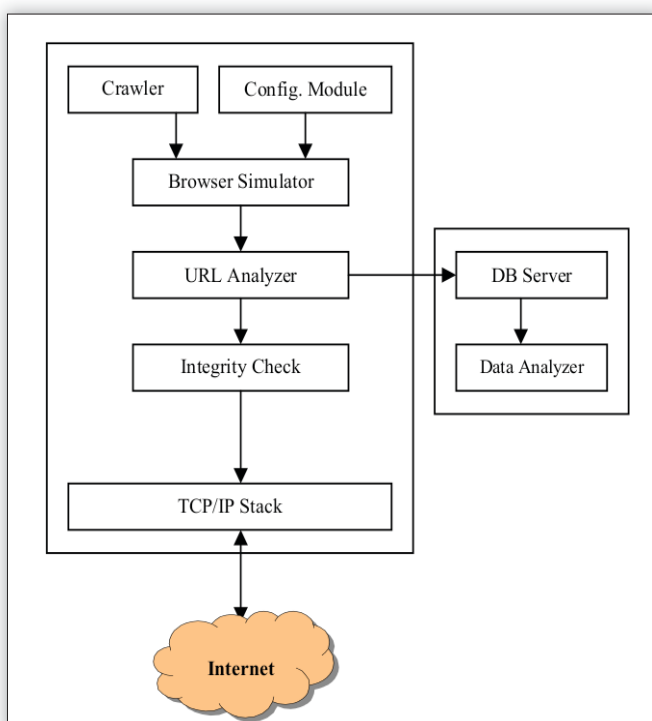


Figure 1.

Human behaviour simulation

One of the main client honeypot goals is to behave like a human. However, low-interaction even high-interaction honeypots do not have features to achieve this purpose completely. Lets look closer into this problem i.e. by dealing with dialog boxes. More websites use typically pop ups with two options: Ok or Cancel. The reactions of the websites can be different, depends on user selection. Malicious websites could make this problem harder, because they implement boxes which have to be filled in by user. In this case, user reaction is necessary to determine the server response. Example of this “fill out dialboxes” is CAPTCHA, which is typically challenge-response test. This CAPTCHA is often used by benign websites as a protection against spam robots. On malicious websites, CAPTCHA allows to hide malicious activity. Current client-side honeypots (SpyBye, HoneyMonkey, Capturem MonekySpider, HoneyC) do not simulate user interaction with dial box. Exploits which need more complex user interaction with website cannot be automated by current implementation of client-side honeypots.

Delayed Exploit

A delayed exploit is a very important issue, if we consider high-interaction honeypot which uses integrity control check mechanism. In some cases, the delay between the initial infection and the complete compromise may occur. This delay occurs only within high-interaction honeypot, because low-interaction honeypots apply directly pattern-matching algorithm on the server response. There are three possible reasons of this delay:

- One of the possible scenarios could be: web pages successfully exploit our system. Then, they download and prepare to install more malicious software on our system. Time which is necessary to detect malicious software on our system is consumed by downloading malware.
- Logic or time bomb. Principle of this bombs relies on exploits which contain malicious web page where the exploit triggers only after a given period of time.
- Exploits triggered by user. Some kind of malicious software needs to be triggered by some interaction with user: clicking mouse, opening some application etc.

Real-Time Integrity Check

Integrity check method can found out whether websites are modified or change something on client system. To find out if something causes changes, we need some time (maybe a couple of minutes). Installed malicious software can hide itself by using various rootkit techniques, and thus make it harder to detect any changes. In another simple word, this integrity control is unreliable. Therefore, the integrity check should be performed in real-time. This integrity control check could be achieved using API hooking, which is interception of API calls that could be used to change the execution flow. Attackers use this method to exploit client systems using rootkits.

Attacks Against Internal Security Policies

Most current implementation of high-interaction honeypots uses integrity control check to detect malicious software. But in other words, they can not detect exploits that do not make any persistent-state changes or make those changes inside

the browser. Attacks against internal policies, like unauthorized access to the history or cookies, might be neglected.

Oday Attacks

This kind of attacks may not be detected using low-interaction client honeypot, as this kind of honeypots use algorithm, which relies on the signatures of known attacks. If this attacks make any change it could be detected using high-interaction honeypots.

Invisibility of client honeypots

Similar to server honeypots, the client-side should have the “invisibility” features too. Invisibility means, that malicious software cannot recognize that the HTTP request is sent by client honeypot. Various techniques can be used to aim this feature.

Anti-crawling techniques

Malicious web pages use fingerprint of client honeypots to detect that request is sent by honeypot. This fingerprinting by malicious web pages is done because client honeypot use automated crawlers. If malicious web pages can detect the honeypot, they can also change they behaviour. This problem is hard to be solve, so client honeypots should behave like a browser as much as possible. Another issue which is related with anti-crawling techniques is: the amount of sent requests. Anti-crawlers can limit amount of requests to one IP address. Solution to this problem lies in using intelligent crawling instead of crawling the whole web site – by looking for suspicious files, scripts and images .

Virtual Environment Detection

Using virtual machines by honeypots is a good practice, because we have ability to easily reset the exploiting machine without influence of our real system. On the other side, the presence of virtual machines can be easily detected using various methods. Virtual machine detection codes can be embedded on the exploited web page. If the malicious web page detects the virtual machines, it can stop triggering the exploit, behave differently, block the honeypot IP or do something to keep it hidden from detection. Thus, honeypots should take a measurements to protect virtual machine from fingerprinting and detections. This ability provide the another technologies called dynamic honeypots.

Geo-location Attacks

Nowadays, it is not a surprise that the user can be geologically located by his IP address. This options use some attackers to compromise visitors in certain country or location. This problem can be handled in two ways:

- Honeypot run across many different networks
- Hiding honeypot behind TOR service proxies

Integration of client honeypots

Until today, there have been few complete client honeypot developments that integrate various detection methods and capabilities of both low and high-interaction honeypots, and are available to the public. Furthermore, no open source client honeypot is coupled with commercial tool to provide real-time security for the end-user. We must keep in mind that honeypot is new technology used to add the value to secure client-side systems. Client-side honeypot faced large web spaces, various web technologies, browser behavior and strong integrations with operating system.

Effectiveness of client honeypots

We can see that honeypots act like computer-human interaction tool and their effectiveness can be defined by accuracy and completeness of client attacks. Given this, we can measure three factors of client honeypots: speed, detection accuracy and invisibility.

Speed

Speed is a significant expressions to describe the ability of the client and can be expressed by number of sites which can be connected and inspected in a certain period of time. If we consider the aim of client honeypots, which is to inspect websites for malware and the size of the Internet, it is clear the client honeypots should have high speed to achieve their goals. This speed influences mostly:

- resources and hardware, network connections
- client honeypot implementation (more complex implementation honeypot, the slower honeypots are)
- detection algorithm play one of the main role too. The pattern-matching method is faster than integrity control check method.

Detection Accuracy

It is clear, that client honeypot should have high accuracy rate. This accuracy can be measured by false positive and false negative rate. In the case of the high-interaction client honeypots, false positive rate can be neglected, thus false negative provides the accuracy of detecting malicious web pages. With low-interaction client honeypots, both false positive and false negative are expected to exist. The ability of client-side honeypots to detect malicious web pages is influenced by honeypots themselves and the operation environment characteristics such as:

- detection algorithm
- delayed exploits (logic bombs etc)
- evasion techniques

Invisibility

The value of honeypots depends on gathered data, but unlike server honeypots, client honeypots do not require to use deception to entice hacker to initiate attack. Client honeypots should be kept undetectable from malicious websites. This ability allows client honeypots gather more data and then identify more attacks.

Dynamics Honeypots

Now we know how different type of honeypots work, and how they can help us. All types of honeypots have one common property: blending with the system. This property aims to facilitate technology called Dynamic Honeypots.

The most critical part of dynamic honeypot is how it learns about our network – what systems do we use and how the systems are being used. One way to do this is to: actively probe the network, determine which systems are alive, type of system and which services they provide. This approach is not very elegant, because we would constantly need to scan out environment to get last system update.

Another approach is to utilize the power of fingerprints. Instead of actively probing the systems, the passive fingerprinting sniffs traffic from the network and analyzes. Sniffed fingerprints can be consulted with database of known fingerprints of

specific systems. This method significantly reduces network bandwidth. This sniffing is continuous – so as an organization network changes, these changes can be captured in real time. This becomes critical for maintenance of honeypot over long term. Major disadvantage of this method is that it works well only in LAN, not in across routed networks. Given this, we can deploy more than one dynamic honeypot, depending on organization size, configuration and network topologies.

Once the honeypot is deployed (as appliance or single box), it spends some time watching and learning the organization of network. By capturing, it finds information about operations systems, which kind of services are offered etc. Once the honeypot learn the environment, it can begin to deploy more honeypots. Power of this kind of honeypot lies in the ability to mirror your environment. Given this, the honeypot seamlessly blend in, making much more difficulty for the attacker to identify it. Moreover, this passive learning does not stop, it continuously monitors the network. Dynamic honeypots really dynamically response to network changes. As we can see dynamic honeypots vastly reduce configuration time and maintain them in constantly changing environment.

Dynamic honeypots can radically help in deployment and maintenance of honeypots. With these features like learning and monitoring network in real time, they become a solution of a future for the honeypots. They not only become cost-effective to deploy and maintain, but they have better integration with organization network.

Conclusion

Client honeypot is pretty new technology that aims to solve the weaknesses of traditional server honeypots and other security tools. This kind of honeypots use two methods to detect client-side attacks. Each approaches has benefits and shortcomings. So far, more sophisticated method has not been developed yet. Some implementation try to combine both approaches to reach best detections method. We have to keep in mind that the current client honeypots are still in the phase of development.

Honeypot dramatically changes the strategy of protection. Rather then passively protect, it allures attacker or even search malicious websites. As it has been mentioned earlier, this powerful concept has many advantages but the best advantage is that honeypot can fend off various techniques which are used by malicious software.

MR. LUDVIK

graduated at [Czech Technical University](#) in 1996. In 2005 he successfully defended his Ph.D. thesis on Data Security in Computer Networks and I was awarded Ph.D. degree. In 2000 he participated on securing the International Monetary Fund conference in Prague. He provides counseling to [Ministry of Informatics Czech Republic](#) and [Czech Data Protection Office](#). He provides also counseling for private sector and among my client are e.g. bank and prestigious legal firms. He teaching on prestige private Czech University and cooperate with University of Žilina. He holds an office of Technical Director in the [4safety, a.s](#) company.

MR. MICHAL SRNEC

graduated at University of Žilina <http://www.fri.uniza.sk> in 2011. From 2011 is postgraduate student on Faculty of informatics science and management department of information networks www.kis.fri.uniza.sk. Works for <http://www.4safety.cz/> as security consultant focusing on secure calling.

DETECTING MALWARE WITH HONEYCLIENTS

DAVIDE CANALI

In this article I'll explain you what an honeyclient is and how honeyclients work, before putting hands on the code to see in practice how to set up one for our needs, using one of the most recent and freely available honeyclient solutions.

This article will first introduce you to the state of the art in the matter of malware detection using honeyclients, showing a short history of honeyclients and the different types of honeyclients on the market. Then, you'll learn how to setup one of the most recent and complete open source honeyclient systems, allowing you to analyze any kind of content (URLs, executable files, PDFs, documents, ...) on a virtual machine running Windows. In order to understand this article, you'll need only some basic knowledge of Linux and of the VirtualBox virtualization solution. A basic knowledge of Python is a plus, even though not necessary.

Honeyclients

Traditionally, when we talk about an honeypot, we imagine a server that has been specifically setup to be attacked, in order to be able to observe and study the moves of the attacker during an eventual compromise of the system.

Another kind of honeypot exists, though: *client honeypots*, aka *honeyclients*, are the client counterpart of classic honeypots. As such, honeyclients are active clients used to test servers that are possibly found responsible for delivering malicious content to visitors.

In most of the cases, honeyclients emulate a browser - in order to detect drive-by-download exploits on the web - or other applications, such as PDF and document readers, often used to infect computers running outdated versions of Windows, MS Word or PDF readers.

Research on honeyclients has recently become quite a hot topic, and the last couple of years have given birth to a wide variety of client honeypot systems, both open- and closed-source.

There are two main categories of client honeypots: *high interaction* and *low interaction*.

A high interaction honeyclient is usually a full system, equipped with real software and with no functional limitations: this type of honeyclient is usually run in a sandboxed environment, so that if an exploit occurs, its side effects can be easily contained. That's why, in general, high interaction honeyclients are run inside virtual machines: apart from containing the attacks, this solution allows to detect exploits just by monitoring state changes on the virtualized operating system. Some well known high interaction honeyclients are the MITRE Hon-

eyClient, Microsoft's HoneyMonkey, CaptureHPC, or Google Safe Browsing's honeyclients. These are all used to detect malicious web pages and run a web browser on a real operating system inside a virtual machine.

Low interaction honeypots, on the other hand, use lightweight clients to interact with servers. In general, these consist in fake applications, developed ad-hoc, simulating the behavior of a real client. In this kind of honeypot the detection of exploits is done through pattern matching and the use of heuristics, as in this case there's no real system to exploit and the attack can not succeed. Projects employing this approach for the detection of exploits on web pages are Wepawet, PhoneyC, JSUnpack, NOZZLE.

Why do these different types of client honeypot exist? Well, because each of these categories has its advantages and disadvantages. The drawback of high-interaction

honeyclients is the fact that the analysis is expensive. Considering that these honeyclients run in a virtual machine, in order to analyze a resource they often need to fetch it and render (in the case of web pages or documents) or execute it (in the case of executables) with the correct client application. Finally, after the analysis of each resource, the virtual machine needs to

be restored, since an exploit could have been triggered and the platform can no longer be trusted. It's easy to understand, then, that the analysis of a single resource with a high interaction honeyclient can take up to several minutes, and there's not much we can do to speed it up, apart from using parallelization (e.g., running several virtual machines at the same time). Another drawback of this kind of honeypots is that, in order to detect an attack, they require the exact vulnerable configuration to be installed on the system, e.g. Windows XP SP 1 equipped with Internet Explorer 7 and Flash Player 3.0. If this is not the case, certain classes of attacks won't be triggered simply because our system is not vulnerable.

Low-interaction honeyclients, on the other hand, have the advantage of not requiring a special configuration in order to detect attacks, as they employ fake, emulated clients that are able to emulate several vulnerable configurations at the same time. Another advantage of this is that, employing fake clients, these systems cannot be exploited, so it's not necessary to re-initialize the environment after analyzing each resource. The

disadvantages of using low interaction honeyclients are that, compared to high interaction ones, they're usually built for the detection of a specific class of attacks, and as such they can have lower detection capabilities especially if used for detecting a broad range of exploits. In addition to this, the analysis is still, usually, quite slow, as emulating one or more real clients at the same time is no easy task.

Existing solutions

As I said in the beginning of this article, many honeyclient projects have been proposed in the last couple of years, and some of them have attracted the attention of academic and industrial security communities. So, if you want to detect malware out there, on web pages, Word documents, or in executable files, chances are that a solution to your problem has already been proposed, and if you're lucky, it may even be open source :)

Let's thus have a look at some of the most known, or recent, honeyclient solutions.

- Capture-HPC, a well known client honeypot developed at the Victoria University of Wellington, NZ, is one of the first client honeypots systems to be published on the web. The analyses are done on VMWare server virtual machines, but the project doesn't seem to be active anymore (the latest release is from 2008)
- The MITRE Honeyclient, a high-interaction honeyclient developed at MITRE. It's a browser based client honeypot used mainly for the detection of malicious websites, and for this purpose it also incorporates a crawler that can be seeded with a list of initial URLs from which to start the analysis.
- HoneyMonkey is a high-interaction honeyclient developed by Microsoft Research for the detection of malicious websites. It is proprietary and not available for download.
- SHELIA is a client honeypot developed mainly for the analysis of emails. It receives emails through Outlook Express, IMAP connections or manual submission and visits every link appearing in the messages, as well as opening/executing all the attachments, in order to detect possible exploits.
- HoneyC, another project developed at the Victoria University of Wellington. It's a low interaction honeypot that emulates a browser visiting an URL; it makes use of static signatures in order to detect attack pages.
- PhoneyC, a low interaction client honeypot that emulates a web browser and uses dynamic analysis to deobfuscate the contents of malicious web pages.
- Wepawet is a framework for the analysis of web based threats. It is based on emulation and can detect exploits on web pages (JavaScript based), PDF files and Flash resources. It's offered as a web service and it's widely used and appreciated in the security community thanks to its detailed reports and ability to deobfuscate malicious code.
- Trigona, a quite recent, VirtualBox-based, high interaction honeyclient, It consists of just a couple of Perl scripts interacting with VirtualBox, and it's not difficult to setup and extend.
- Cuckoo Sandbox, a very recent and still active project, proposing a modular solution for the analysis of malware. Mainly written in Python, and (currently) based on VirtualBox, this framework has a modular and easily extensible architecture allowing users to create their own analysis

modules and scripts for parsing the results.

In the following sections, I'll explain you how to set up Cuckoo to build your own honeyclient for the analysis of malicious web pages.

Among all the available projects, my choice of Cuckoo is due to its high customizability and ease of use, advantages that I haven't found, to the same extent, in several other projects I personally tried. Finally (and quite surprisingly in this field, I would say :P), Cuckoo has also a well written documentation, which is always a good news and saves users from a lot of unnecessary headaches.

As stated on the homepage of the Cuckoo project (<http://www.cuckoobox.org>), Cuckoo Sandbox is a malware analysis system allowing the user to retrieve:

- Trace of performed relevant win32 API calls
- Dump of network traffic generated during analysis
- Creation of screenshots taken during analysis
- Dump of files created, deleted and downloaded by the malware during analysis
- Trace of assembly instructions executed by malware process

Of course, as I already said, Cuckoo is very versatile and easy to extend, so nothing will stop you from using it for analyzing whatever kind of resource you want!

Setting up Cuckoo

Let's then proceed to the installation. For this, we need: a Linux operating system, a recent release of VirtualBox and the latest stable release of Cuckoo. On VirtualBox we also need to set up a Windows virtual machine, so a Windows installation CD might come in handy.

In the following, I'll take Ubuntu Linux as reference operating system, since this is the most common end-user distro out there and it's the one on which Cuckoo has been tested the most. In case you use a different GNU/Linux distribution, some of the commands may vary.

Installing and configuring the Guest OS

First, we need to install VirtualBox and set up a proper Windows virtual machine for running the experiments. I won't cover the basic steps in detail, so please refer to VirtualBox documentation for more details in case you need some help.

If you don't have it already, download and install VirtualBox, either downloading it from <https://www.virtualbox.org/wiki/Downloads> or via synaptic/apt/aptitutde.

Create a new virtual machine and install Windows on it. If you can, I suggest you to install an old version of Windows, such as Windows XP SP3 or previous, as these versions are more vulnerable to exploits than the latest ones. As for the virtual machine options (RAM, hard disk space, etc.) you can choose anything you want. I called my virtual machine cuckoo.

Once you have your VM set up, install the VirtualBox Guest Additions on it (once the VM is running, on its window select *Devices* → *Install Guest Additions...*)

For Cuckoo to execute and control the analysis, we need Python 2.6 or greater installed on Windows. So, from the guest OS point your browser to <http://python.org/download/> and download the most recent Python 2.X installation package. Install it.

Optionally, if you want your Cuckoo setup to take screenshots and dump assembly instructions, you need to install

the Python Image Library (PIL) and WinAppDbg, respectively. At this point, your guest OS is ready to execute malware. However, in order to trigger the majority of exploits, we need to install the client software we expect to be exploited, and to tune some system settings. In this article, since we want to analyze URLs, we'll make sure to have an old version of Internet Explorer, an old Firefox release and an old version of Adobe Reader (to analyze PDFs we might find online).

These are just suggestions, and you can install any app you want based on what you want to analyze. For malicious web pages, Internet Explorer 6 shipped by default on every Windows XP SP3 release is a great choice, as it's full of security holes! If you're seriously concerned about the detection capabilities of your Cuckoo box, then I suggest you to search and install client applications based on the amount of exploits against them, or at least the number and severity of CVEs published for them. You may also want to set up multiple Windows virtual machines to run with Cuckoo, each one with a different configuration, so that an exploit that doesn't get triggered by the first VM might be triggered by the second one.

So, let's make sure we have IE 6 installed, and let's find and install old releases of Firefox and of Adobe Reader.

Note: finding old software releases is not always an easy task! For security and compatibility reasons, many software companies discontinue old releases of their applications, and make them not available for download. In this case, websites like oldversion.com and oldapps.com can be useful.

Finally, make sure the software update feature is disabled for every software we install: an automatic update during the analysis might interfere with our results, and generate unwanted disk and network activity that will be unnecessarily logged. The same advice should be followed on the operating system side, so make sure you disabled the Windows Firewall and the Automatic Update feature before continuing. Another advice is to set, for all the applications you intend to use in your honeyclient, all the security settings to their lowest allowed values, or to completely disable them. As an example, in the browsers you plan to use, disable all the security settings, enable ActiveX controls, Java applets, and accept all kinds of content. This will increase the chances your honeyclient will be hit by an attack. In the following picture you can see an example of setting such properties on IE 6.

At this point, we've almost set up everything on the guest. Before finalizing the VM setup, we need to install Cuckoo and prepare the environment needed for guest OS and host OS to communicate.

Setting up Cuckoo

Before installing Cuckoo, we need to install Python and some Python libraries. Python 2.6 or 2.7 are the current preferred versions.

```
$ sudo apt-get install python python-magic python-dpkt
python-mako
```

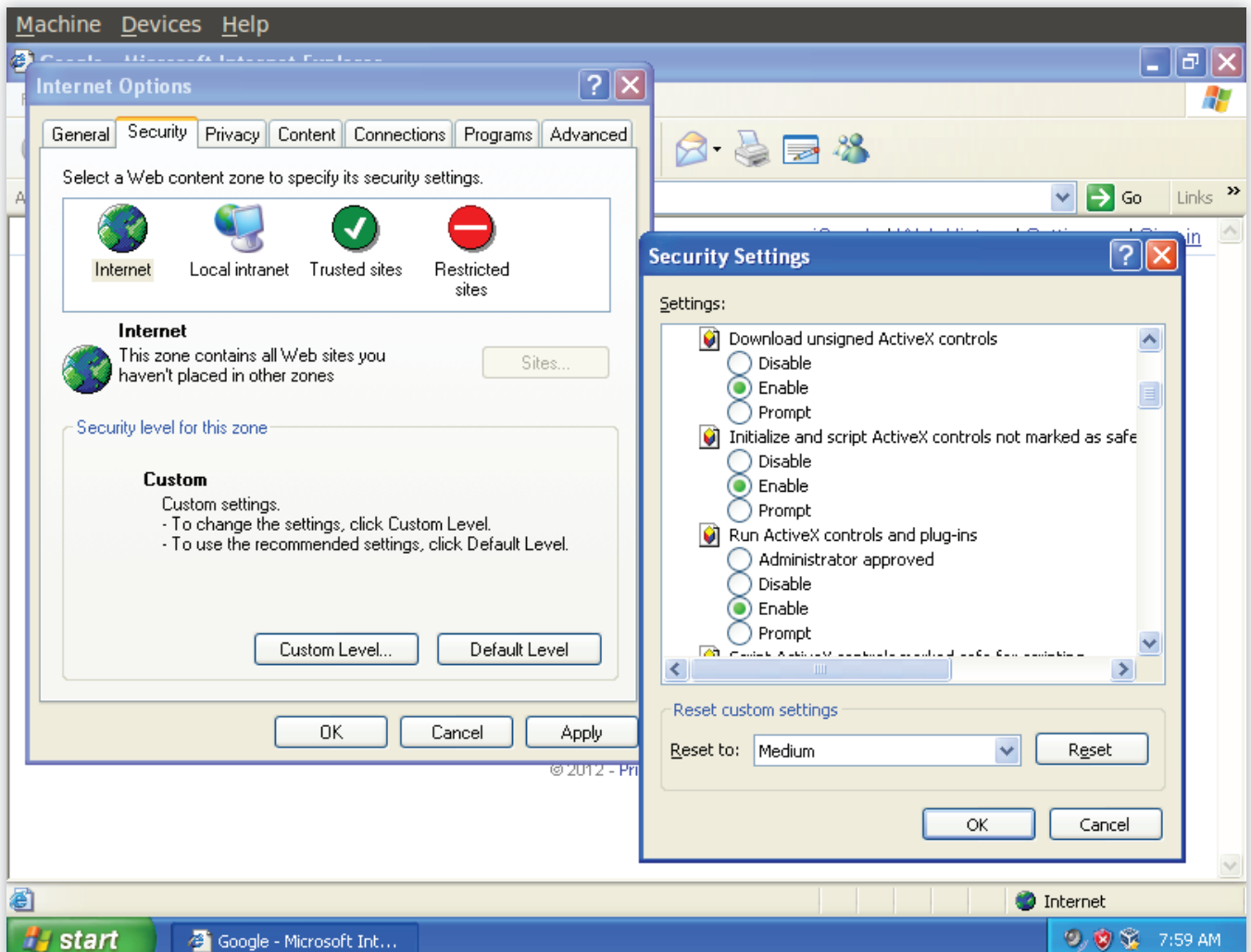


Figure 1: setting custom security settings in Internet Explorer

```

www.cuckoobox.org
Copyright (C) 2010-2012

[2012-02-09 17:12:24,371] [Core.Init] INFO: Started.
[2012-02-09 17:12:24,402] [VirtualMachine.Check] INFO: Your VirtualBox version is: "4.0.12", good!
[2012-02-09 17:12:24,403] [Core.Init] INFO: Populating virtual machines pool...
[2012-02-09 17:12:27,742] [VirtualMachine.Restore] INFO: Virtual machine "cuckoo" successfully restored to current snapshot.
[2012-02-09 17:12:28,076] [VirtualMachine.Infos] INFO: Virtual machine "cuckoo" information:
[2012-02-09 17:12:28,076] [VirtualMachine.Infos] INFO: \_| Name: cuckoo
[2012-02-09 17:12:28,077] [VirtualMachine.Infos] INFO: | ID: 1cd62d90-c8ff-4da3-89a7-cce6b928b37b
[2012-02-09 17:12:28,077] [VirtualMachine.Infos] INFO: | CPU Count: 1 Core/s
[2012-02-09 17:12:28,077] [VirtualMachine.Infos] INFO: | Memory Size: 512 MB
[2012-02-09 17:12:28,078] [VirtualMachine.Infos] INFO: | VRAM Size: 18 MB
[2012-02-09 17:12:28,078] [VirtualMachine.Infos] INFO: | State: Saved
[2012-02-09 17:12:28,079] [VirtualMachine.Infos] INFO: | Current Snapshot: "first snapshot"
[2012-02-09 17:12:28,079] [VirtualMachine.Infos] INFO: | MAC Address: 08:00:27:55:FB:FF
[2012-02-09 17:12:28,079] [Core.Init] INFO: 1 virtual machine/s added to pool.
[2012-02-09 17:12:28,217] [Database.Init] INFO: Generated database "db/cuckoo.db" which didn't exist before.

```

Figure 2: first execution of Cuckoo

Download Cuckoo from its website: <http://www.cuckoobox.org/>

The current stable release is 0.3.2; however, being the project recent and very active, you can expect new releases to be rolled out quite frequently. You can also download the latest development stage from Cuckoo's git repository, but I would not recommend it for a production usage.

Once you have the cuckoo package, choose its installation directory and uncompress it to this destination:

```
$ tar -xzf cuckoo_0.3.2.tar.gz -C /cuckoo_install_dir
```

Now, let's create a cuckoo user: in order to allow it to control VirtualBox, we need to add it to the VirtualBox users' group (by default, it is 'vboxusers' – check in your /etc/group file if such a group exists).

```
$ sudo adduser cuckoo
$ sudo adduser cuckoo vboxusers
```

As a side note, whenever you add a new user for an application, it's always a good practice to make sure it does not have the permission of logging in remotely on the system. E.g., if you have ssh installed, you should add it to the DenyUsers list in the sshd_config file, or set its login shell to /bin/false

That's it. The installation is completed. Before starting to analyze malware, though, we need to customize Cuckoo's configuration, in order to tell it, at least, where to store the analysis results, which virtual machine to run, and few other details.

Configuration

Once inside Cuckoo's installation directory, move to the sub-directory conf/.

The first file we find here is cuckoo.conf: this is the main configuration file of the system, and controls its general behavior and analysis options. Luckily for us, every option is well

documented so feel free to modify it at your own will. I'll just change and comment a couple of options:

```
analysis_timeout = 120
watchdog_timeout = 300
```

These options set, respectively, the default duration of the analysis (2 minutes), and the maximum time span during which Cuckoo should wait for the guest OS to finish its operation (5 minutes). If the watchdog_timeout is hit, this usually means the VM or the analyzer are stuck and need to be killed.

The choice of good timeouts depends on several factors: since, once a malicious binary is run, the exploit can be triggered at any point in time, nothing prevents malware writers to wait some seconds, if not minutes or hours, for the real infection to begin. That's why a longer analysis time usually allows for higher detection rates; on the other hand, though, this means that our system will be slower and will analyze a lower number of samples per day.

```
sniffer = off
```

In this case, we leave this option as is, as we haven't set up tcpdump for sniffing network traffic from the VM. However, this does not mean we won't have a network trace, but simply indicates that we'll use VirtualBox's built-in network tracing capability. If you want to use tcpdump, refer to Cuckoo's documentation in order to set it up for sniffing the virtual machines' traffic.

```
engine = VirtualBox
enabled = cuckool
mode = gui
python = C:\Python27\python.exe
```

Listing 1: the code of the cuckoo_run() function inside the ie.py analysis package

```
def cuckoo_run(target_path):
    config = ConfigParser.ConfigParser()
    config.read(target_path)
    url = config.get("InternetShortcut", "URL")
    pids = []
    internet_explorer = "C:\\Program Files\\Internet Explorer\\iexplore.exe"
    suspended = False
    (pid, h_thread) = cuckoo_execute(internet_explorer, url, suspended)
    cuckoo_monitor(pid, h_thread, suspended)
    pids.append(pid)
    return pids
```

Listing 2: the code of our custom url.py URL analysis package

```
import os
import sys
import ConfigParser
import urllib2
sys.path.append("\\\\VBOXSVR\\setup\\lib\\")
from cuckoo.execute import cuckoo_execute
from cuckoo.monitor import cuckoo_monitor
TEMPFILE='C:\\temp\\temporary.pdf'
def download(url):
    try:
        uh = urllib2.urlopen(url)
        resp = uh.read()
        uh.close()
        tempfile = open(TEMPFILE, 'wb')
        tempfile.write(resp)
        tempfile.close()
    except Exception:
        return False
    return True
def cuckoo_run(target_path):
    config = ConfigParser.ConfigParser()
    config.read(target_path)
    url = config.get("InternetShortcut", "URL")
    pids = []
    #adjust these paths to match your own configuration
    internet_explorer = "C:\\Program Files\\Internet Explorer\\iexplore.exe"
    acrobat = "C:\\Program Files\\Adobe\\Reader 8.0\\Reader\\AcroRd32.exe"
    firefox = "C:\\Program Files\\Mozilla Firefox\\firefox.exe"
    suspended = False
    #very basic check to see if the URL is possibly a PDF.
    #If so, launch adobe reader; otherwise, IE and FF
    if (url.lower().endswith('.pdf')):
        #download and read with acrobat
        if download(url):
            (pid, h_thread) = cuckoo_execute(acrobat, TEMPFILE, suspended)
            cuckoo_monitor(pid, h_thread, suspended)
            pids.append(pid)
        else:
            #download failed: nothing to analyze
            pass
        else:
            #Firefox
            (pid, h_thread) = cuckoo_execute(firefox, "%s" % (url), suspended)
            cuckoo_monitor(pid, h_thread, suspended)
            pids.append(pid)
            #IE
            (pid, h_thread) = cuckoo_execute(internet_explorer, "%s" % (url), suspended)
            cuckoo_monitor(pid, h_thread, suspended)
            pids.append(pid)
    return pids
def cuckoo_check():
    return True
def cuckoo_finish():
    return True
```

The first line indicates that we're going to use VirtualBox as virtualization engine. So far, this is the only supported choice.

The second line sets Cuckoo's internal name(s) of the virtual machines we want it to run. In case we want multiple VMs, a comma separated list of strings has to be supplied.

mode has to be set to 'gui' if we want Cuckoo to show the GUI of the VM while the analysis is running, or to 'headless' if we don't want it to spawn the graphical interface. For the moment, I set this value to 'gui' for testing purposes, but once your malware analysis system is correctly set up you may want to set this value to 'headless' so that the analysis can

run in background or on systems not having a graphical user interface.

Finally, the last line has to be set to the path in which python has been installed (on the Windows virtual machine).

For each name you specified in the enabled comma separated list, then, you need to create a section named as the Cuckoo's internal name of the VM. In this case, we only have one machine called cuckoo1:

```
[cuckoo1]
name = cuckoo
```

```
username = admin
password = cuckoo
share = shares/cuckool
```

name is the name we used to register the virtual machine in VirtualBox. Remember, also, to set username and password to the credentials used on your Windows account, for each VM you want to run with Cuckoo, since they're required for the host to run commands on the guest OS. Finally, share is the shared folder that will be used to exchange data between host and guest OS. You can choose any path you want for it, but the last directory must have the same name as Cuckoo's current machine ID (the name in square brackets). Before launching the first analysis, make sure this directory exists (all paths, if relative, are relative to cuckoo's main installation directory).

The other configuration file we're interested in is reporting.conf. This file contains switches that enable, respectively, reports to be generated in json format (useful to be exported and parsed by different applications), txt or HTML format. All the formats are enabled by default, and you can set them on or off singularly, depending on your needs.

Putting it all together

Now that we have set up both Cuckoo and VirtualBox, we can put them at work together.

Before launching our first analysis, we still need to modify a couple of details on our virtual machine. Go back to VirtualBox, and add two shared folders: their location on the host

is already configured, in the shares/ directory inside Cuckoo's installation folder.

Let's first power off the VM:

```
$ VBoxManage controlvm "cuckoo" poweroff
Then:
$ VBoxManage sharedfolder add "cuckoo" --name "setup"
--hostpath "/cuckoo_install_dir/shares/setup" --readonly
$ VBoxManage sharedfolder add "cuckoo"
--name "cuckool" --hostpath "/cuckoo_install_dir/
shares/cuckool"
```

The first folder will be shared in read only mode, and will contain the analysis scripts we will launch on the guest. The second one will instead be used for moving data from the guest to the host after the analysis, that why it has not to be mounted in read only mode. If you prefer, shared folders can be also configured via the GUI.

We proceed, then, to enable VirtualBox's builtin network tracing capabilities on cuckoo:

```
$ VBoxManage modifyvm "cuckoo" --nictrace1 on
--nictracefile1 /cuckoo_install_dir/shares/cuckool/dump.pcap
```

This configures the virtual machine named cuckoo to dump all its network traffic on the path specified (which, as you can see, is on the virtual machine's writable shared folder, but is not necessary to store it there). Finally, we are ready to save our machine's state into a snapshot, that will be restored at



Figure 3: Cuckoo running our custom analysis. You can see that both Firefox and Internet Explorer are in execution.

the beginning of each analysis. Before taking the snapshot, restart the VM and let it boot completely. Once the machine is ready, type:

```
$ VBoxManage snapshot "cuckoo" take "first snapshot" --pause
```

This will put the machine in a paused state, and take a snapshot of it. Once this is done, you can close the VM and restore the snapshot:

```
$ VBoxManage controlvm "cuckoo" poweroff
$ VBoxManage snapshot "cuckoo" restorecurrent
```

Or, do the same using the GUI.

Analysis and customization

At this point, we can finally launch Cuckoo:

```
$ ./cuckoo.py
```

So far, nothing exciting happens. Cuckoo is just waiting for us to submit a file or a URL to analyze.

Let's open another terminal and try to submit a URL:

```
$ ./submit.py -u "http://www.google.com/"
```

We get a message saying that we didn't specify any package, so Cuckoo is going to use the default Internet Explorer package. This makes us understand that Cuckoo provides different analysis packages. In fact, several packages are provided, and they'll be automatically selected depending on the kind of resource we submit or chosen manually by us at the time of submission. Available packages are: exe, dll, pdf, doc, php, ie, firefox, tracer. The use of a package can be forced calling `submit.py -p package_name`

Since in this article I propose you to analyze web pages, we'll always submit URLs via the `-u` option of `submit.py`. However, this script offers several possibilities of submitting resources to Cuckoo, the most general of which is `./submit.py` filepath (in this case Cuckoo automatically detects the kind of resource and spawns the analysis using the package considered the most appropriate). In order to see all the options available for submission, type:

```
$ ./submit.py --help
```

There are also other ways to submit files to Cuckoo: interacting with its SQLite database and using Cuckoo's python library functions, but I won't cover them in this article. You're encouraged to read Cuckoo's documentation for details about them.

If the submission went fine, the terminal on which we launched cuckoo should inform us that cuckoo is starting in "gui" mode and we should see the virtual machine GUI popping up and opening Internet Explorer. After, approximately, the number of seconds we set as `analysis_timeout`, the virtual machine will be powered off. Cuckoo will then inform us that the results of the analysis are available:

```
(Task #1) [Core.Analysis.SaveResults] INFO: Analysis results
        successfully saved to "analysis/1".
```

At this point, the next job in queue, if any, will be started, restoring the snapshot and analyzing its resource.

Let's check the contents of the folder in which the analysis task we just ran saved its results. There will be at least 3 folders: `files/` (containing all the files that were created or deleted during the analysis), `logs/` (containing behavioral traces of the executed system calls), `reports/` (containing the reports in the different formats we enabled in the `reports.conf` configuration file), plus `shots/` and `trace/` if the Python Image Library and WinAppDbg were installed, respectively.

Apart from this, the folder should contain a `dump.pcap` network trace that you can analyze with your favorite packet analyzing tool (e.g., Wireshark, tshark, ...) and where, if you ever analyze a drive-by-download web page and you're lucky enough, you will find unexpected connections to redirection, exploit and Command and Control servers.

A very nice feature of Cuckoo is, finally, its web interface. You can launch it running the script `web.py`:

```
$ ./web.py
```

This will create a web server listening on localhost, port 8080, to which we can point our browser and see the results of our analyses in a nice and user-friendly format (if `reporthtml` is on in `reporting.conf`).

Customizing the analysis

We want to finish setting up our honeyclient for the detection of web pages hosting malware. A nice feature of our system would be that of being able to analyze an URL with *more than one browser* (or different versions of the same) so that the chances the exploit will be triggered are higher.

I mentioned that Cuckoo is highly customizable, and writing your own analysis package is quite easy. In this last section, so, I'll guide you step-by-step into creating our own analysis module, allowing us to launch the analysis of a target URL on two different browsers in parallel (in this case, Internet Explorer and Firefox). This way, we'll be able to detect web pages containing exploits that target either one of the two browsers. And, since the web is full of (malicious) PDF documents and it would be a pity to miss them, we'll also write a couple of lines of code to handle the situation in which the URL we want to analyze points to a PDF.

For more details on how to write your own analysis packages, please refer to the exhaustive documentation provided on Cuckoo's website.

Before starting, make sure to have both Firefox and Acrobat Reader installed on your virtual machine, and of taking a new snapshot once these apps are installed and configured.

Let's take a look at how an analysis package looks like. Open the file `shares/setup/packages/ie.py`

This file is responsible for analyzing URLs launching Internet Explorer. Three functions are defined: `cuckoo_run()`, `cuckoo_check()` and `cuckoo_finish()`. They're respectively called at the beginning of the analysis, every second during the analysis, and at the end of the analysis.

Let's have a deeper look at the (very short and simple) `cuckoo_run()`: The first four lines can be a bit confusing. Anyway, keep in mind that we are analyzing an URL, and this URL has to be passed to the VM for analysis. In order to do this, Cuckoo actually writes this URL into a file, which is then passed to the guest OS via the shared folder mechanism. The first 3 lines of the function are used to read the URL from the file.

Note: when analyzing a regular file, instead, the parameter `target_path` of `cuckoo_run()` can be used directly as a regular



path and doesn't need any special processing. Have a look at other packages, such as *pdf.py* or *exe.py* to see an example of this.

In the following lines of code, the variable `internet_explorer` is set to the path of Internet Explorer's binary, then the browser is launched calling `cuckoo_execute()`. This function accepts, as parameters, the path to the executable, the URL to be analyzed (in general, any arguments to pass to the application), and a flag that has to be set to true if we want the process to be created in suspended state. It returns the process PID and a handle to the thread.

Finally, `cuckoo_monitor()` is executed. This is probably Cuckoo's framework most important function, as it injects and starts monitoring the process we just created, allowing to collect its system call trace.

What we want, then can be coded easily in a few more lines of code. Let's take *ie.py* as a stub, so copy it, give it the name you want and place it in `shares/setup/packages/`. I called it *url.py*; its code is very similar to the original *ie.py* and contains only some additions to the `cuckoo_run()` function.

I just added a very naïve check on the filename, to try understand if the URL we're going to analyze is a PDF document. If it's not, both Firefox and Internet Explorer will be executed to visit the same URL.

If, instead, the URL appears to end in `.pdf`, we'll consider it a PDF document. In order to analyze it, we first need to download the file to a temporary location (because Adobe Reader doesn't open URLs directly) and then spawn and monitor an old version of Adobe Reader trying to read it. The download code is contained in the `download()` function and should be self explanatory.

We're finally ready to use our client honeypot :)

Let's launch it on the website of the magazine, using our new custom analysis package:

```
$ ./submit.py -p url -u 'http://hakin9.org/'
```

Here is a screenshot of Cuckoo running with our custom analysis package, called "url".

And, finally, Cuckoo's web interface showing the results of this analysis.

Conclusion

In the first part of the article, after explaining what an honeyclient is, I gave an overview of the current state of the art in the matter of malware analysis using honeyclients. In the second part of the article, using a hands-on approach, I introduced the reader to the recent Cuckoo project, an easily extensible and modular malware analysis system, and accompanied him through the process of installing and configuring it in order to build a customized client honeypot for the analysis of web pages.

DAVIDE CANALI

is currently a Ph.D. Student at EURECOM (France) and member of iSecLab, the International Secure Systems Lab. He holds a BSc. and a MSc. in Computer Engineering, both from Università di Bologna, and has been active in the field of web security for more than 5 years. His research interests include Web and Network Security and Malware Analysis. Prior to joining EURECOM, Davide worked one year as intern at the Computer Security Lab of University of California, Santa Barbara.

HAKIN9 EXTRA

EXCLUSIVE INTERVIEW

WITH FRED COHEN

Fred Cohen is best known for coining the term “computer virus”, he is also the founding father of majority of known virus defense techniques. Mr. Cohen significantly contributed to the development of digital forensic evidence examination. The keyword here is: “deception”, as it has always been used in Cohen’s research within the field of information protection (the invention of Deception Toolkit). Chronologically speaking, in the 70s, he devised network protocols for secure digital networks carrying: voice, video and data. In the 80s, he came up with integrity mechanisms for secure operating systems and in 1984, he presented his abstract on computer viruses which had major impact on IT-Security field. Dr. Cohen worked as a consultant for major companies, he is an inventor, a teacher, a researcher and an analyst – a genuine IT-Security’s jack of all trades but his capabilities and ambitions reach far beyond that. Currently, Fred Cohen is the President of California Sciences Institute and CEO of Fred Cohen & Associates. Apart from working for the government and industry, Dr. Cohen contributes to non-profit organizations and undertakings to make sure his knowledge serves the right cause. He holds a Ph.D. in Electrical Engineering from the University of Southern California (1986). Source: <http://all.net/resume/bio.html>



of DTK was that it only needs to be changed by altering the scripts for services, and not by changing the code itself. After that, we went to the new and improved approach of the various now-patented technologies including things like the one detailed in the deception area above under “2002: Method and Apparatus Providing Deception and/or Altered Operation in Information Systems”

PLC’s (programmable logic controllers) and DCS’ (distributed control systems) aren’t known for their robust security practices and implementations. You were active in

PCCIP, what trends are you seeing in that critical infra-structures?

PCCIP ended at the end of the Clinton era, but I am still active. The main trend is that we are putting weaker systems in control of more critical assets, producing increased risk with only minimal benefits. At the same time, we are connecting it all together through the Internet to further weaken the infrastructures and increase unnecessary interdependencies. But I can manage the traffic lights from your iPad on the beach!

Have you done any work in mobile space (phones, tablets)? What are your thoughts on security on mobile platforms?

As an approximation, take PCs from the late 1990s, make them smaller, better connected, and less controllable by the IT department and users, add in the improvements in attacker skills, capabilities, motives, etc. but not the defense technology advances, and that’s the security situation with most mobile devices today.

Considering the advanced hacks and APT’s nowadays, honeypots and honeynets need to change as well. What is your view on this topic and what will the future bring when it comes to honeypots/nets to keep up with these kind of threats?

Deception technology doesn’t really need to improve so much as to be properly applied. Honeynets/pots are not really much more than the birds in mineshafts of old. They were never intended to be defenses per se - just sensors. Deception ToolKit was intended to do more than most honeypots (i.e., to disrupt attack methods and tools, not just detect them), but the follow-on technology in deception is good enough so that it is practically undetectable by attackers if properly managed and operated.

In the computer field, you are well-known for your coining the term “viruses”, your defense techniques and your work in honeypots. Your experiments and your papers helped set the way for many security researchers. For those not familiar with your work would you mind describing your initial virus experiment?

They are detailed at <http://all.net/> -> Research -> Technical Safeguards -> 1984: Computer Viruses - Theory and Experiments and in other papers in that collection.

Can you describe the idea behind your Deception Toolkit and port 365?

They are detailed at <http://all.net/> -> Research -> Deception for Protection AND under <http://all.net/dtk/index.html>

Since DTK’s first release in 1999, how has DTK changed to match current trends and attacks?

I think it was first released on November of 1998. The last official update was in <http://all.net/dtk/v1999-08-18.html> The idea

What do you believe to be the involvement of government and military structures in computer virus development today?

I don't know what (if) they are doing (it) in secret, but I imagine they are (governments in general).

How many signatures/algorithms do you think a computer should have to detect malware?

0/1

What do you believe to be one of the best ways to protect information?

First off, protecting information is not the goal of information protection. The definition of protection is 'keeping from harm'. That is, keeping people (and other creatures that feel pain and pleasure, live and die, etc.) from being harmed (information doesn't feel pain and is not harmed when altered). But harm associated with information (symbolic representations in the most general sense) can come in many forms and to many parties. You might want to direct your readers toward <http://all.net/> -> "2012-01-31: Influence Operations" as well.

I don't believe there is one best way. Information protection is a complex issue involving many equities. One person's attack is another person's intelligence operation. Is it protecting information in the form of financial records to not aggressively break into the systems of those who attack those same records? When you are attacked (whatever that may be) should you not be able to aggressively defend?

Would you consider yourself the father of viruses?

I describe my role as "the person who defined the term "computer virus" and the inventor of most of the widely used computer virus defense techniques, " I also did most of the early scientific work in the field and published most of the peer reviewed papers in scientific journals in the early years (pre-1993).

What challenges did you endure while your programming skills were in their infancy?

Programming skills are not particularly a problem for me. I was writing programs in the 1960s. I learned Pitt Interpretive language (PIL) in grade school, was a systems admin and learned PDP somethings (punched paper tape - way better than card decks) and other minicomputer languages and systems in the 1960s, then lisp and APL (IBM mainframe), then PDP8 assembler (and hardware) in high school by reading the hardware manual, then basic, then microcode, and so forth. As I learned electrical engineering in college and graduate school, I tended to write software by defining finite state automata and using the available language to express them. I only ever had one actual course in (computer) programming and it was a comparative languages course in maybe 1974. By then I was already a night operator of PDP10s, one of the early IMPs in the ARPAnet, and so forth. I think our ARPAnet node was number 8 or some such thing.

In my experience, the hard part is figuring out what to program, not programming it.

What was your first person/homebrew computer?

In the early 1960s (maybe when I was 7 or 8 years old) I built a mechanical computer. I guess that's homebrew enough. I liked the radio shack TRS80 and related computers when they came out - far more usable and really consumer products. Storage on audio tape - not so good...

What would you like your legacy to be within the field of security?

I'm not that old yet. My view when I was young was that the only important things in life are the things that last. I viewed (and still view) computer viruses as the first truly artificial life form on Earth (who knows what happened somewhere else in the Universe when - and time is relative...). I thought, and still do, that reproducing symbol sequences with the potential to evolve in general ways are a life form, and can find no definition of life that isn't purely directed toward ruling them out, that disagrees with this. I also believe that they have all the potential for good and evil of any other advancement in science and/or engineering. That has nothing to do with security, but I think it is pretty interesting and worthwhile.

I view my body of work in information protection (I prefer that term - keeping from harm by symbolic representations) as just that. A body of work. I enjoy it, I am interested in it, and I think there is a lot to do. To the extent that I can help bring forward a more humane and positive future for people as we enter the information age, I certainly would like to do so, but I am unsure of how much effect a single person really has. Computer viruses were, I think, bound to happen. The conditions were right and getting more so. The fertile field cannot remain unseeded except in a sterile environment. The real question is how the field evolves depending on how it is seeded. I hope that I seed things in a direction that benefits humanity in the long run, but it's hard to tell the future. I believe that more thoughtful people will tend to do better, but perhaps what's more important is people that care about other people. If you combine caring with thoughtfulness, I think you do better still. I hope that my work helps people become more caring and thoughtful.

Any pointers for anyone starting out in the security field

Start by reading everything at <http://all.net/> (it's all free) and all of the references and their references and their references. Work hard, study hard, and play hard. It takes all 3. Recognize going in that there are people involved and work on your understanding of the human cognitive system and condition. Seek to build a science and engineering discipline - and call snake oil what it is. You can't do your job if you are worried about losing your job.

Consider very seriously your personal ethics. You can do a lot of harm and a lot of good in this field. Try to work toward freedom and justice for all, even if the path forward is not always easy to see and no known solution is perfect.

Always remember, you are fallible. "Ain't a horse that can't be rode, ain't a man that can't be throwed"

and finally Star Wars or Star Trek?

I like them both (prefer early Star Trek over later Star Trek, Episode 4 and 5 to the others in Star Wars) - and Battlestar Galactica - and 2001 - and a very long list of other outstanding science fiction going back to Jules Verne. I have a collection of perhaps a few thousand of them - old time radio as well as TV. The Twilight Zone, the original Mission Impossible series, etc.

Thanks again for your time and your contribution to the security field.

Pleasure.

interview by Nick Baronian

**Keeping You At the
Top of Your Game.**
Paper • eBooks • Dropbox

Pragmatic
Bookshelf



www.pragprog.com



WE @ iViZ

HATE

FALSE POSITIVES

AS MUCH AS YOU DO

1st Cloud based Application
Penetration Testing service

No Tools | No Consultants | Zero False Positives | Cost effective



www.ivizsecurity.com