

# **Mitigating scanners and crackers**

White Paper

Renaud Bidou

renaudb@radware.com

**November 2007**

**Version 1.0**

## Contents

<b>INTRODUCTION</b>	<b>6</b>
<b>SCANNERS AND CRACKERS</b>	<b>6</b>
THE NEED FOR A PROTECTION	6
MITIGATING THE THREAT	6
<b>MAPPING THE THREAT</b>	<b>7</b>
<b>CLASSIFICATION AXIS</b>	<b>7</b>
<b>DISCOVERY AND IDENTIFICATION TOOLS</b>	<b>7</b>
NETWORK LAYER	7
OPERATING SYSTEM LAYER	8
APPLICATION LAYER	8
SCRIPTS	9
USERS	9
<b>VULNERABILITY TESTING TOOLS</b>	<b>10</b>
GENERIC SCANNERS	10
DEDICATED SCANNERS	10
ATTACK TOOLS	11
<b>MASS GENERATOR</b>	<b>12</b>
FUZZERS	12
BRUTE FORCERS	12
<b>THREATS MAP</b>	<b>13</b>
<b>DETAILS OF OPERATIONS</b>	<b>14</b>
<b>DISCOVERY TOOLS</b>	<b>14</b>
LAYER 2 DISCOVERY TOOLS	14
LAYER 3 DISCOVERY TOOLS	16
LAYER 4 DISCOVERY TOOLS	18
<b>IDENTIFICATION TOOLS</b>	<b>22</b>
OPERATING SYSTEMS IDENTIFICATION TOOLS	22
APPLICATION IDENTIFICATION TOOL	28
<b>CRACKING TOOLS</b>	<b>33</b>
VULNERABILITY SCANNERS	33
MASS GENERATORS AND ATTACK TOOLS	37

<b>MITIGATING THE THREAT</b>	<b>42</b>
<b>THE CHALLENGE OF MITIGATION</b>	<b>42</b>
AXIS OF RESEARCH	42
THE NEED FOR NEW TECHNOLOGIES	42
COMMON PREVENTION ISSUES	42
<b>DETECTION METRICS</b>	<b>43</b>
GLOBAL NETWORKING ACTIVITIES	43
DETECTING ERRORS	44
TUNING ERROR STATISTICS	46
<b>CHARACTERIZATION AND BLOCKING</b>	<b>48</b>
COMMON METHODS AND LIMITATIONS	48
FOOTPRINTING THE THREAT	49
<b>CONCLUSION</b>	<b>54</b>
<b>SCOPE OF MITIGATION</b>	<b>54</b>
<b>FUTURE WORK AND IMPROVEMENTS</b>	<b>55</b>
<b>APPENDICES</b>	<b>56</b>
<b>APPENDIX A : REFERENCES</b>	<b>56</b>
DOCUMENTS	56
STANDARDS	56
TOOLS	56
<b>APPENDIX B : COMMAND LINES</b>	<b>57</b>
LAYER 2 SCANS	57
LAYER 3 SCANS	57
LAYER 4 SCANS	57
OS SCANS	58
APPLICATION SCANS	58
VULNERABILITY SCANS	58

## Table of figures

Figure 1 : Threats map	13
Figure 2 : Layer 2 scan with THCrut	15
Figure 3 : Layer 2 scan capture	15
Figure 4 : Passive Layer 2 scan with arpswatch	16
Figure 5 : ICMP Netmask scan with THCrut	18
Figure 6 : ICMP Netmask scan capture	18
Figure 7 : Nmap FIN scan	21
Figure 8 : Nmap FIN scan capture	21
Figure 9 : Nmap OS scan capture	23
Figure 10 : Xprobe run	25
Figure 11 : THCrut run with fingerprint details	26
Figure 12 : sinfp scan capture	26
Figure 13 : amap triggers for http	29
Figure 14 : amap responses for some http servers	29
Figure 15 : nmap run for application identification	29
Figure 16 : hmap run against Apache 2	31
Figure 17 : hmap run against IIS 5.0	32
Figure 18 : extract of server.db	34
Figure 19 : server-msg.db sample entry	35
Figure 20 : Attack point detail	36
Figure 21 : SQLiX run	39
Figure 22 : SQL injection obfuscated by SQLiX	39
Figure 23 : SQL injection response with valid SQL	40
Figure 24 : SQL injection response with invalid SQL	41
Figure 25 : Legitimate user connection profile	47
Figure 26 : Malicious source connection profile	47
Figure 27 : Threats map coverage	54

## Index of tables

Table 1 : Layer 2 scanners metrics	15
Table 2 : Layer 3 scanners metrics	17
Table 3 : Layer 4 vertical port scan metrics	20
Table 4 : OS scan protocol distribution	27
Table 5 : OS scan rate based characteristics	27
Table 6 : Generic application scanners metrics	29
Table 7 : HTTP fingerprint samples	31
Table 8 : HTTP identification scanners metrics	32
Table 9 : Vulnerability scanners metrics	36
Table 10 : Global networking activities	43
Table 11 : Erroneous responses	44
Table 12 : Layer 3 footprint characteristics	49
Table 13 : Layer 4 footprint characteristics	50
Table 14 : Application layer footprint characteristics	51

## **Introduction**

### **Scanners and crackers**

Scanners and crackers are the main tools used for security testing automation. In the hands of experts they are usually used to speed up security audit processes.

Either they will be used to perform low added value operations such as network sweeps or port scans, or they will make it possible to quickly perform tests that would need months if they were done manually – like password cracking.

They can also be considered as a knowledge base, which will simplify the research for vulnerability exposure of specific operating systems or applications.

### **The need for a protection**

However, malicious individuals can also take advantage of such tools in order to quickly and efficiently find vulnerabilities in target systems and network infrastructures. Moreover, worms usually propagate via automated testing and infection process, imitating (or simply copying) the technology used in scanners and crackers.

Therefore being able to block such behavior becomes mandatory as it will eliminate most large-scale hacking attempts, block worms and considerably slow down targeted cracking operations.

### **Mitigating the threat**

Understanding how such tools work is a mandatory step before designing a mechanism that would efficiently block their operations. Therefore we will first detail their operational mode and try to find with criteria that could be leveraged to provide an effective protection.

Once those criteria will be identified we will analyze the feasibility to implement a practical solution for production networks.

## Mapping the threat

### Classification axis

There are many tools used to automated security tests. However, classification can be made simple as long as we consider two axes: the functional and the application scopes.

The functional scope will have tools being marked as member of one of the following categories :

- Discovery and identification;
- Vulnerability testing;
- Mass generators.

The second axis will classify tools according to their operational layer: network, operating system, application (which stands for standards software), scripts (custom settings and developments) and users.

This last axis is trivial and tools will naturally be affected to the appropriate category. Therefore we will focus on the first one and build a tools map starting from this axis.

### Discovery and identification tools

Discovery and identification tools are designed to automate the process of mapping target infrastructure. From THCrut [thcrut] that will generate ARP requests on a local network to httpprint [httpprint] that will try to fingerprint web servers, these tools represent the largest category.

However, distinguishing the application scope is an easy task.

#### Network layer

This is the most popular category as operations to be performed are quite basic. Therefore these tools are easy to write, analyze and copy; and most automated malwares such as worms or bots do have their own, home-made, discovery tool. However, and as they all rely on the same techniques, the way to mitigate their operations remains the same.

Major families of network discovery tools are :

- ARP discovery tools: They are only efficient on local networks and detects system which are up by performing simple ARP requests for IP addresses of a subnet. Main tools implementing this technique are THCrut [thcrut] and ettercap [ettercap];
- ICMP discovery tools: They perform different kind of ICMP requests and analyze answers they get. While most basic tools, such as fping [fping] will quickly generate ICMP echo requests, more advanced scanners, like nmap [nmap], will make it possible to use stealthier requests, using ICMP timestamp (code 13) or even ICMP address mask (code 17 – specified in RFC 950 [rfc950]). This last

## Mitigating Scanners and Crackers

option is less reliable as most OS don't answer, but definitively faster and more discreet;

- Port scanning tools: There are many ways to scan ports. Most of them have been described as early as 1997 in "the art of scanning" [artofscanning] and are very well known. They have been implemented in hundreds of tools. Most famous are probably nmap [nmap] and hping [hping], but most malicious implementations in malwares are simply coded from scratch.

Being able to block operations similar to those per

pdo 427(C718(321958P0898(o)8.46521(o).4666583(l)24(65

## Mitigating Scanners and Crackers

testing (FTP servers don't understand the same commands as Web servers) and fingerprinting.

Another classification criteria is the applicative scope coverage provided by those tools. Some of those tools are going to simply focus on one application (httpprint focuses on web servers, svmap [svmap] focuses on SIP servers); while some other will have a larger scope, such as (surprisingly !) scanssh [scanssh], or amap [amap].

### Scripts

The script layer basically includes all custom parameters, settings, and any kind of development that can be done on top of legacy applications. Therefore a CGI script a graphical front-end or even the directory structure of a web server belongs to this layer.

As each and every application that runs at this layer is different, discovery and identification is the most important step to be taken. Once these operations are performed, malicious users or software are provided with enough information to launch generic attacks based (such as cookie tampering, SQL injection, etc.) on specific parameters.

The very specific nature of each script-layer component naturally implies that discovery tools are dedicated to one type of protocol or application. However, the level of details and techniques used are highly variable. As an example nikto [nikto], wikto [wikto] and NTOinsight [ntotools] both have HTTP discovery and identification capabilities :

- nikto will only look for common administrative pages and users home directories;
- wikto will try a huge list of directories and common pages (backend scanning), will follow hyperlinks to scan the whole web site (spidering) and even request google search engine for indexed pages that belong to the server (googleing);
- NTOinsight will mainly spider the remote site, however, it will provide far more information such as visible and hidden variable, cookies etc.

Moreover, layer seven protocols are getting more and more generic and are used to transport different type of application data. This is typically the case of HTTP, which is widely used to transport XMLRPC commands to web services. In this special case discovery techniques are different, and implemented in special tools, such as SIFTwms [siftwms].

### Users

Last the enumeration of users defined on a system is a necessary step when the objective of the attack is to take advantage of other users rights, or simply to impersonate further illegitimate operations.

Hopefully most common applications make quite impossible to scan for existing users or to dump users list. It is then a question of either finding a file mistakenly left of the server or performs advanced timing attacks. Anyway, except in the case of files found thanks to web crawling or backend scanning, there is no automated way to find such list.

However, for some new protocols and application it is common to observe the resurgence of old mistakes. As an example, SIP components that don't implement authentication (which is optional), will generate a message when a requested user does

## Mitigating Scanners and Crackers

not exist. Therefore users discovery is just a question of scanning, based on a dictionary. This is what SIPSscan [sipsca] simply does.

### Vulnerability testing tools

Discovering the network, identifying OS and mapping applications is a first step, but not enough to compromise an IT infrastructure. Therefore the next step is to evaluate the exposure of target systems to vulnerabilities. This is what vulnerabilities scanners are made for.

They can be classified in three families:

- Generic scanners : they will perform thousands of tests and provide a list of potential vulnerabilities that may be exploited;
- Dedicated scanners : which will also test for multiple vulnerabilities but only affecting one specific type of OS or application.
- Exploitation tools : they launch real attacks toward targeted systems.

Malicious users and some bots scanning engine will perform a generic or a dedicated scan first, while most worms will immediately try to exploit the vulnerability, would the target be vulnerable or not.

#### Generic scanners

Generic scanners have to quickly test thousands of potential vulnerabilities against multiple targets. Most of these tests belong to one of the two following categories:

- Accurate identification of the targeted application (type, nature, version etc.) then search into a vulnerability database. In this schema, the efficiency of the scanner depends on the identification accuracy and the update of the vulnerability database;
- Stimuli generation and analysis of the reaction, that should be different if the application is vulnerable or not. Accuracy is then a factor of the way the reaction to a stimulus is analyzed.

As an example the response to a IIS Unicode exploit should be a HTTP code 200. However some web application firewalls will answer a HTTP code 200 but with a different content. Therefore an appropriate analysis should look for content with "Directory of", which does only matches English versions of Windows...

One of the most obvious consequences of this operation mode is that generic scanners either have their own identification tool (nikto [nikto] uses its home-made mechanism); or rely on external ones (sometimes it is just an option, as for nessus that can use amap results).

#### Dedicated scanners

Designing a generic scanner is not necessarily difficult. However maintaining it means making sure that databases, third-party tools, local settings, fingerprints etc. are up to date. This task needs a huge investment in terms of time for research and development. And this is quite impossible to achieve when a tool is designed by individuals for non-commercial purpose.

## Mitigating Scanners and Crackers

Therefore many tools have a limited scope, either in terms of protocols, attacks or application types. Operation modes are similar to those of generic scanners, but the focus on very specific targets makes them usually more efficient and more accurate.

The most famous tool of this category is probably nikto [nikto], which is targeting web servers. It implements more than 3.000 security checks, a scanner for hidden pages and miscellaneous evasion techniques in order to bypass IDS and IPS. Another example of an even more focused tool is iisvs [iisvs]. This tool tests for more than 1.700 vulnerabilities on IIS v5.0 servers.

Another popular field of application for dedicated scanners is SQL injections. In this case the issue is more the huge number of database types, parameters combinations, attack vectors and evasion techniques; it clearly makes it impossible to manually scan for vulnerabilities and it requires a very dedicated expertise and even though most tools will only focus few servers and techniques. A good example is SQLiX [sqlix], that tries to exploit vulnerabilities leading to the control of MS-SQL and MySQL databases, through 4 different vectors : error messages, comment injections (only for MySQL), blind statement injection and blind string injection. On top of it implements quite subtle obfuscation techniques, making it impossible to detect with a signature.

Usually this kind of scanner is used by malicious users who already know which application they want to focus. In such case the scanner will go through most filtering and will not generate many alerts from firewalls. This increased stealth is of great value in such case.

### **Attack tools**

In this last category we find the tools that are made to exploit vulnerabilities. This should be the last stage, once the application has been discovered and identified, and once vulnerability has been found.

The most famous tool that belongs to this category is the Metasploit framework [metasploit]. It makes it possible to launch attacks with different payloads which effects range from the execution of a command to the spawn of a VNC server on the victim system. Once again some more specialized tools exist, in order to facilitate the exploitation of specific vulnerabilities. SQLNinja [sqlninja] is one that gets more and more popular as it makes the exploitation of SQL injections on MS SQLServer databases trivial and very powerful.

Exploitation of network weaknesses can also be simplified. Many tools, such as Yersinia [yersinia] or ettercap [ettercap] implements functions to bypass network-based security, interfere with traffic or even hijack network control mechanisms such as DHCP. As usual some more targeted tools exist and focus specific network functions and devices. A good example of these is voiphopper [voiphopper] which targets network devices to perform VLAN hopping.

However powerful are those tools, they usually require a minimum knowledge and understanding of what is going on. Therefore their use is rarely automated and mostly the fact of malicious and more or less skillful users.

## Mass generator

Mass generators are tools designed to launch a massive number of similar operations at high speed. Most common programs that belong to this category are password brute forcers and fuzzers.

### Fuzzers

In this last case, tools are used to extensively test input handling of applications. Therefore they will generate several types, values and size of inputs and check the behavior of the remote system. These tools are essentially used for vulnerability research and are not part of the threats we should consider here.

### Brute forcers

From a certain point of view brute forcers are members of the fuzzers' family, as they are going to test multiple authentication credentials until a special behavior (authentication) happens. Whatever families they belong to, brute forcers are quite popular, usually trivial to implement and to launch. Their efficiency relies on two main factors:

1. Quality of the dictionary: quality does not necessarily mean quantity. There are not millions of common and default passwords. Moreover it is important to fit to contextual parameters, such as the native language of users. And, last but not least, testing useless or unlikely passwords is a loss of time, and impacts the parameter below.
2. Speed and reliability: even with an optimized dictionary, there will be thousands of login / password combinations to test. This will lead to two main consequences. First test may be very long if the brute forcer is not fast enough. The need for speed brings to the second consequence, which is a potential lack of reliability when techniques used to improve speed are not well implemented. To many threads, open sockets or even IPC may impact the stability of the hosting OS, and lead to crashes, freezes or information loss, generating "false-positive" results.

Generic brute forcers will make it possible to target multiple applications. The most famous of these tools is Hydra [hydra] from the TCH group. It is able to test passwords for more than 20 authentication types, from more usual one (http, ftp) to quite exotic ones (cvs, pcanynwhere, teamspeak etc.). This kind of tools test authentication methods defined in standards, such as the Basic HTTP authentication.

However, for application-specific authentication, like the authentication form that can be found on most web sites, special brute-forcers must be used. Brutus AET2 [brutus] is the state of the art for web-based applications. But even more specific tools can be found for special purposes. As an example Venom [venom] has been designed to brute-force windows authentication via the WMI (Windows Management Instrumentation) interface, bypassing a potential account lock.

Of course the choice of passwords that respects minimum security requirements, such as not being found in a dictionary (...) will noticeably reduce the efficiency of such tools.

# Mitigating Scanners and Crackers

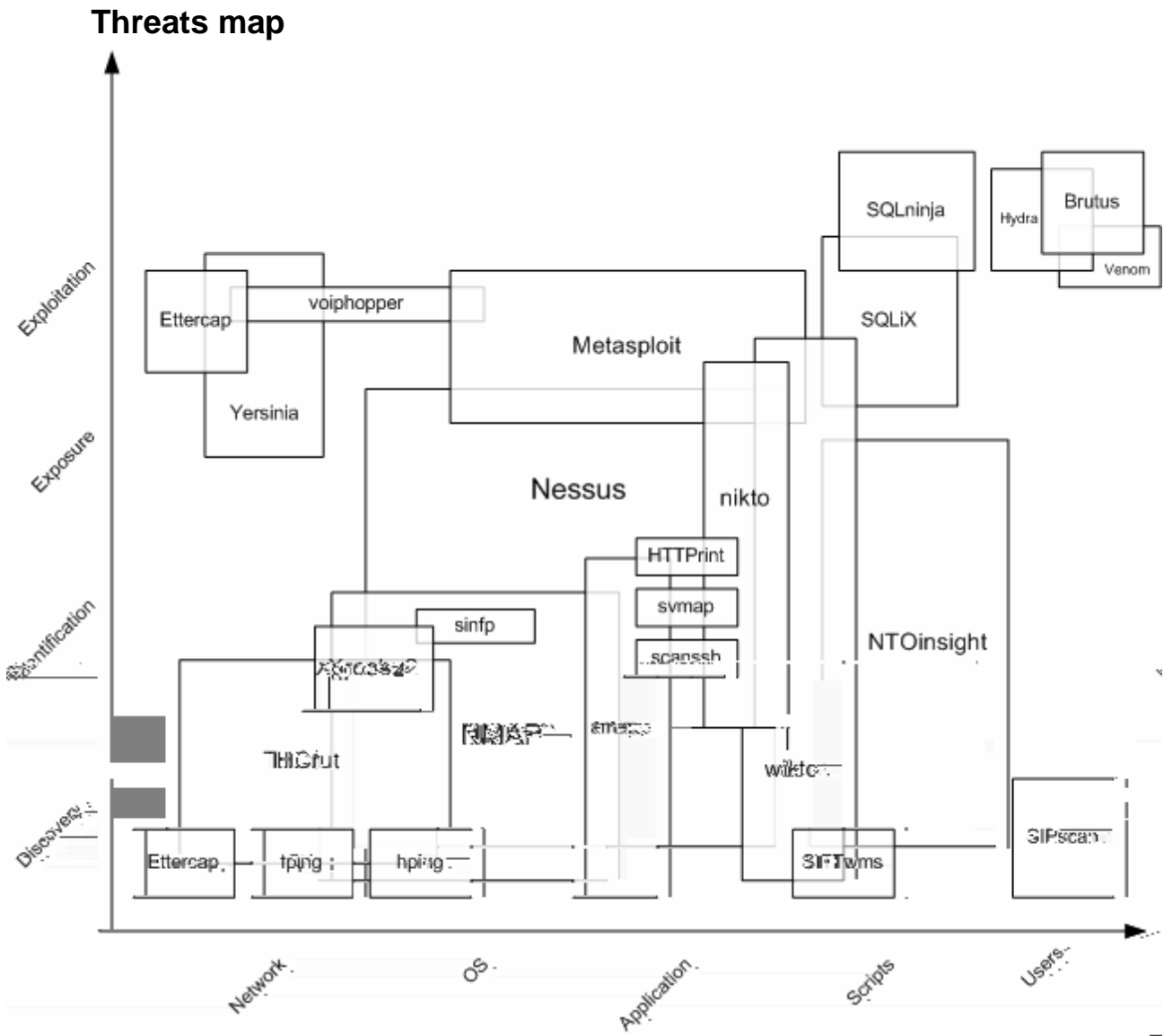


Figure 1 : Threats map

## Details of operations

Once the main threats have identified, it becomes necessary to understand how they proceed to perform their tasks.

This is the only appropriate way to design a security technology.

### Discovery tools

Blocking the discovery process is one of the most important step to be taken to block automated attack tools. As these tools apply predefined procedures stopping the very first one usually stops the whole process.

In the meantime many exploitation tools need information about the targeted system, from the IP address to the parameters to be injected. Without the appropriate knowledge the attacker best scenario is a blind test that will require many attempts, with a success rate dramatically lowered.

Network discovery includes probes that will try to get layer 2 (MAC) to layer 4 (transport) information from networked devices.

#### Layer 2 discovery tools

At layer 2, discovery is based on ARP requests and responses. Of course this kind of discovery is limited to the local network of the attacker.

There are two techniques, used by tools to discover systems at layer 2 : active and passive scan.

- **Active scan**

The first one is an active technique that simply sends out ARP requests to each and every IP address of the local network of the interface. Any running system will answer these requests and provide the information to the attacker.

Below is an example of a scan launched with THCrut [thcrut] and an extract of the tcpdump capture performed during the scan. In this capture the “responding” exchanges have been highlighted.

```
[root@localhost audits]# thcrut arp 192.168.205.1-192.168.205.254
thcrut: using source ip 192.168.205.163
thcrut: listening on eth0
192.168.205.1    00:02:a5:46:22:25 Farallon Computing Inc
192.168.205.254 08:00:37:18:5e:4c Ericsson Business Comm.
192.168.205.11  00:02:a5:ea:92:a6 Farallon Computing Inc
192.168.205.200 00:d0:58:61:1d:c0 CISCO SYSTEMS, INC.
192.168.205.140 00:01:02:da:83:d0 INFORMATION TECHNOLOGY LIMITED
192.168.205.147 00:19:b9:2a:83:1c COMTRON, INC.
192.168.205.154 00:18:8b:b3:1f:19 OCTAGON SYSTEMS CORP.
192.168.205.102 00:13:ce:73:a9:db DIGITAL EQUIPMENT CORPORATION
192.168.205.174 00:11:11:41:58:1a Cicada Semiconductor, Inc.
```

## Mitigating Scanners and Crackers

```

192.168.205.50 00:10:f3:0a:c0:6a NEXCOM INTERNATIONAL CO., LTD.
192.168.205.182 00:01:e6:16:6b:80 Madge
192.168.205.185 00:06:5b:b8:17:64 TRI-DATA Systems Inc. Netway products, 3274
192.168.205.60 00:03:b2:2b:63:00 Acer Counterpoint
192.168.205.249 00:b0:64:62:6c:c0 Cisco Systems, Inc.
192.168.205.250 00:05:32:23:71:c0 LECTRA SYSTEMES SA
192.168.205.251 00:17:9a:af:b7:62 SOURCE-COMM CORP.
192.168.205.252 00:17:9a:af:b7:6a SOURCE-COMM CORP.
24 packets received by filter, 0 packets dropped by kernel

```

**Figure 2 : Layer 2 scan with THCrut**

```

[root@localhost ~]# tcpdump arp and host 192.168.205.163
10:14:50.532041 arp who-has 192.168.205.1 (Broadcast) tell 192.168.205.163
10:14:50.532191 arp reply 192.168.205.1 is-at 00:02:a5:46:22:25 (oui Unknown)
10:14:50.540617 arp who-has 192.168.205.64 (Broadcast) tell 192.168.205.163
10:14:50.552671 arp who-has 192.168.205.127 (Broadcast) tell 192.168.205.163
10:14:50.564644 arp who-has 192.168.205.190 (Broadcast) tell 192.168.205.163
10:14:50.572614 arp who-has 192.168.205.253 (Broadcast) tell 192.168.205.163
10:14:50.584614 arp who-has 192.168.205.2 (Broadcast) tell 192.168.205.163
10:14:50.592614 arp who-has 192.168.205.65 (Broadcast) tell 192.168.205.163
10:14:50.604618 arp who-has 192.168.205.128 (Broadcast) tell 192.168.205.163
10:14:50.612616 arp who-has 192.168.205.191 (Broadcast) tell 192.168.205.163
10:14:50.624617 arp who-has 192.168.205.11 (Broadcast) tell 192.168.205.163
10:14:50.624880 arp reply 192.168.205.11 is-at 00:02:a5:ea:92:a6 (oui Unknown)
10:14:50.632691 arp who-has 192.168.205.3 (Broadcast) tell 192.168.205.163
10:14:50.644618 arp who-has 192.168.205.66 (Broadcast) tell 192.168.205.163

```

**Figure 3 : Layer 2 scan capture**

The only metric that can be analyzed for this kind of attack is the rate of requests sent by the discovery tool. We compared the results obtained by THCrut and Ettercap [ettercap]. Results are given in the table below.

**Table 1 : Layer 2 scanners metrics**

	THCrut	Ettercap
Network Size	255	
Duration (s)	8,43	3,11
Packets Sent	712	255
Packets Received	25	24
PPS	87,4	89,5

## Mitigating Scanners and Crackers

Hosts Discovered	25	24
------------------	----	----

Although both tools have approximately the same packet rate THCrut uses more probes as it tests up to three times non-responding systems. Therefore the scan is definitively more visible from the network point of view, but also more reliable, as THCrut discovered one more host than Ettercap.

- **Passive Scans**

Technology behind passive scans is even more trivial. It relies on sniffing the network and grab ARP requests and gratuitous announces which are broadcasted over the network.

These techniques take longer than active scan, depending on the activity of the network, but are totally invisible from the network. An example of logs provided by arpwatch [arpwatch] is shown in the figure below.

```
Sep  5 08:50:44 localhost arpwatch: new station 192.168.205.1 0:2:a5:46:22:25
Sep  5 08:50:44 localhost arpwatch: new station 192.168.205.154 0:18:8b:b3:1f:19
Sep  5 08:50:46 localhost arpwatch: new station 192.168.205.60 0:3:b2:2b:63:0
Sep  5 08:50:49 localhost arpwatch: new station 192.168.205.11 0:2:a5:ea:92:a6
Sep  5 08:50:49 localhost arpwatch: new station 192.168.205.163 0:f:1f:bd:27:ab
Sep  5 08:50:55 localhost arpwatch: new station 192.168.205.185 0:6:5b:b8:17:64
Sep  5 08:50:58 localhost arpwatch: new station 192.168.205.141 0:50:b6:40:3c:e8
Sep  5 08:51:00 localhost arpwatch: new station 192.168.205.147 0:19:b9:2a:83:1c
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.166 0:19:d2:6c:dd:33
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.121 0:18:8b:b6:c2:24
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.250 0:5:32:23:71:c0
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.174 0:11:11:41:58:1a
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.130 0:1:2:da:83:d0
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.136 0:1:2:da:83:d0
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.135 0:1:2:da:83:d0
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.200 0:d0:58:61:1d:c0
Sep  5 08:51:23 localhost arpwatch: new station 192.168.205.252 0:17:9a:af:b7:6a
Sep  5 08:51:24 localhost arpwatch: new station 192.168.205.182 0:1:e6:16:6b:80
Sep  5 08:51:24 localhost arpwatch: new station 192.168.205.132 0:1:2:da:83:d0
Sep  5 08:51:24 localhost arpwatch: new station 192.168.205.140 0:1:2:da:83:d0
Sep  5 08:51:24 localhost arpwatch: new station 192.168.205.146 0:19:b9:29:d6:5c
```

**Figure 4 : Passive Layer 2 scan with arpwatch**

### Layer 3 discovery tools

Discovering hosts at the network layer is a question of knowing if a host at a specific IP address answers Layer 3 probes. Most basic probes are pings (ICMP ECHO requests). But many alternative ICMP request can be performed, such as ICMP TIMESTAMP or

## Mitigating Scanners and Crackers

ICMP ADDRESS MASK requests. For the purpose of discovering routers the ICMP ROUTER SOLICITATION request can also be used.

However, results will differ, first because all IP stacks don't necessarily answer to all ICMP requests, and second because of possible filtering of ICMP packets. Usually ICMP TIMESTAMP requests will be handled, while ICMP ADDRESS MASK requests are only handled by network devices (and theoretically only router). This is the same for the ROUTER SOLICITATION request, although the support of this request is rarely implemented, mostly for security reasons.

The table below provides main metrics of scans launched with fping [fping], THCrut [thcrut] and nmap [nmap] and implementing different scanning techniques.

**Table 2 : Layer 3 scanners metrics**

	ECHO			MASK		TSTAMP	
	fping	thcrut	nmap	thcrut	nmap	nmap	
Network Size	254						
Duration (s)	41,78	5,99	4,31	5,98	4,01	4,31	
Packets Sent	977	736	498	750	506	498	
Packets Received	REPLY	13	13	14	6	6	14
	UNREACHABLE	44	9	7	9	7	7
	TOTAL	57	22	21	15	13	21
PPS	25	126	120	127	129	120	
Address scanned	254	254	256	254	256	256	
Number of scans	4	3	2	3	2	2	
Host discovered	13	13	13	6	6	13	

All tested scanners launch probes several times during a scan. Nmap will scan the network twice, THCrut three times and fping four times. Nmap has one specific behavior as it will scan for the broadcast and network address and therefore generate additional probes, and usually receive one additional response from the broadcast address.

Below is an example of a scan launched with THCrut [thcrut] and an extract of the tcpdump capture performed during the scan. In this capture the "responding" exchanges have been highlighted.

```
[root@localhost ~]# thcrut icmp -A 10.0.0.1-10.0.0.255
thcrut: listening on eth0
10.0.0.21      icmp_seq=0 ttl=063 mask=255.255.255.0
10.0.0.22      icmp_seq=0 ttl=063 mask=255.255.255.0
10.0.0.31      icmp_seq=0 ttl=063 mask=255.255.255.0
10.0.0.33      icmp_seq=0 ttl=063 mask=255.255.255.0
10.0.0.34      icmp_seq=0 ttl=063 mask=255.255.255.0
769 packets received by filter, 0 packets dropped by kernel
```

**Figure 5 : ICMP Netmask scan with THCrut**

```
[root@localhost ~]# tcpdump proto 1 and host 192.168.205.163
11:01:17.203688 IP 192.168.205.163 > 10.0.0.146: ICMP address mask request
11:01:17.207688 IP 192.168.205.163 > 10.0.0.209: ICMP address mask request
11:01:17.211689 IP 192.168.205.163 > 10.0.0.21: ICMP address mask request
11:01:17.213266 IP 10.0.0.21 > 192.168.205.163: ICMP address mask is 0xffffffff00
11:01:17.215700 IP 192.168.205.163 > 10.0.0.84: ICMP address mask request
11:01:17.223688 IP 192.168.205.163 > 10.0.0.147: ICMP address mask request
11:01:17.227690 IP 192.168.205.163 > 10.0.0.210: ICMP address mask request
11:01:17.231689 IP 192.168.205.163 > 10.0.0.22: ICMP address mask request
11:01:17.233382 IP 10.0.0.22 > 192.168.205.163: ICMP address mask is 0xffffffff00
11:01:17.235694 IP 192.168.205.163 > 10.0.0.85: ICMP address mask request
11:01:17.243692 IP 192.168.205.163 > 10.0.0.148: ICMP address mask request
11:01:17.247690 IP 192.168.205.163 > 10.0.0.211: ICMP address mask request
11:01:17.251690 IP 192.168.205.163 > 10.0.0.23: ICMP address mask request
11:01:17.259690 IP 192.168.205.163 > 10.0.0.86: ICMP address mask request
```

**Figure 6 : ICMP Netmask scan capture**

### Layer 4 discovery tools

These discovery tools are very common and usually known as port scanners or port sweepers. In the first case they will test for a list of open ports on a single host, and such operation is often referred as “vertical” scanning. In the second case the same port is going to be tested on several hosts. One will then use the expression “horizontal” scanning, or sweeps.

There are four techniques to scan for TCP ports and only one for UDP. These techniques are well known and present the following characteristics:

- **TCP Connect:** This portscan is the most simple and makes use of simple TCP connection attempts to the target port. If the connection is refused (RST) the port is considered as closed. If we get no response to the SYN, the port is considered as “filtered”, if the synchronization attempt is acknowledged (SYN-ACK) then the port is considered as opened. The session is completely established, then cleanly closed.
- **TCP Half-scan:** this techniques is similar to the TCP connect to the exception that if the response from the server is a SYN-ACK then the session is immediately reset, before it gets established.
- **TCP FIN:** TCP FIN scan is based on the fact that most TCP/IP stacks do not behave properly when they receive a FIN packet on an open port. Theoretically they should send a RST for any FIN packet that is received on an open port and that doesn't belong to an existing session. However, they usually don't send any notice to the source, while a RST is sent when the FIN packet reaches a closed port.

## Mitigating Scanners and Crackers

- TCP Anomalies: Anomalies-based scans are sending abnormal or illegitimate packets, such as SYN/RST, "Christmas trees" (all flags sets), packet with no flag etc. Scan results are obtained thanks to the fact that in some cases open and closed ports don't behave the same way, some will send out a RST some will not. More subtle differences can be found also, such as the TCP window size that may be set to zero in some cases.
- UDP scans: The only way to scan UDP ports is to send UDP packets to the target port and analyze any response. If the response received is an ICMP port unreachable, the port is closed. If a UDP packet is received the port is probably opened and we got a response from our probe. Otherwise, if no response is received the port is either opened (but the targeted application didn't reply) or it is filtered by a firewall.

Some variants exist, using fragmentation, relays or even bounces, but from the target point of view the behavior will remain the same.

In order to characterize portscanning activities main metrics are number and type of packets generated. We analyzed scans from nmap [nmap] and hping [hping] both vertically on low TCP ports (1-1024).

## Mitigating Scanners and Crackers

**Table 3 : Layer 4 vertical port scan metrics**

		Connect	SYN		FIN		XMAS	
		nmap	nmap	hping	nmap	hping	nmap	Hping
Ports scanned		1024						
Duration (s)		1,25	1,22	4,097	2,45	9,28	1,23	9,28
Packets sent	<i>SYN / FIN</i>	1056	1025	1025	1040	1066	1030	1066
	<i>ACK</i>	6	NA	NA	NA	NA	NA	NA
	<i>RST</i>	6	6	6	NA	NA	NA	NA
	TOTAL	1068	1031	1031	1040	1066	1030	1066
Packets received	<i>SYN / ACK</i>	6	6	6	NA	NA	NA	NA
	<i>RST</i>	1018	1018	1018	1018	1018	1018	1018
	TOTAL	1024	1024	1024	1018	1018	1018	1018
PPS		1667	1673	501	839	224	1662	224
Port discovered		6	6	6	6	6	6	6

Hping default behavior doesn't provide appropriate results as the rate used for packet generation is too high. Therefore a special option (-i u10) has to be applied. Moreover, default behavior when no response is received (after a FIN scan on an open port as an example) is to retry 7 additional times. This behavior explains the 1066 packets generated by hping during FIN and XMAS scans.

In the same situation (FIN or XMAS scan), nmap replays once the scan for non responding (ope,) ports. However,, nmap looks less reliable as the number of packets generated is more or less random. This means that packets are more often considered as lost, even in the case of "connect" scans.

Below is an example of a FIN scan launched with nmap and an extract of the tcpdump capture performed during the scan. In this capture the positives responses (means no reply for the target) have been highlighted.

```
[root@localhost ~]# nmap -sF -p 1-1024 10.0.0.101

Starting Nmap 4.20 ( http://insecure.org ) at 2007-10-18 09:24 CEST
Interesting ports on 10.0.0.101:
Not shown: 1018 closed ports
PORT      STATE      SERVICE
22/tcp    open|filtered ssh
80/tcp    open|filtered http
111/tcp   open|filtered rpcbind
113/tcp   open|filtered auth
199/tcp   open|filtered smux
443/tcp   open|filtered https
```

## Mitigating Scanners and Crackers

```
Nmap finished: 1 IP address (1 host up) scanned in 1.392 seconds
```

**Figure 7 : Nmap FIN scan**

```
[root@localhost ~]# tcpdump host 10.0.0.101 and host 192.168.205.163
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
09:26:53.057502 IP 192.168.205.163.45356 > 10.0.0.101.auth: F
09:26:53.057667 IP 192.168.205.163.45356 > 10.0.0.101.114: F
09:26:53.057816 IP 192.168.205.163.45356 > 10.0.0.101.poppassd: F
09:26:53.057960 IP 192.168.205.163.45356 > 10.0.0.101.mcidas: F
09:26:53.058048 IP 10.0.0.101.114 > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.058196 IP 192.168.205.163.45356 > 10.0.0.101.pop3: F
09:26:53.058321 IP 10.0.0.101.poppassd > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.058424 IP 10.0.0.101.mcidas > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.058569 IP 192.168.205.163.45356 > 10.0.0.101.sunrpc: F
09:26:53.058649 IP 10.0.0.101.pop3 > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.058771 IP 192.168.205.163.45356 > 10.0.0.101.snagas: F
09:26:53.058912 IP 192.168.205.163.45356 > 10.0.0.101.csnet-ns: F
09:26:53.059054 IP 192.168.205.163.45356 > 10.0.0.101.rtelnet: F
09:26:53.059194 IP 192.168.205.163.45356 > 10.0.0.101.sftp: F
09:26:53.059242 IP 10.0.0.101.snagas > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.059409 IP 10.0.0.101.csnet-ns > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.059561 IP 10.0.0.101.rtelnet > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.059689 IP 10.0.0.101.sftp > 192.168.205.163.45356: R 0:0(0) ack
09:26:53.061406 IP 192.168.205.163.45356 > 10.0.0.101.pop2: F
09:26:53.061770 IP 10.0.0.101.pop2 > 192.168.205.163.45356: R 0:0(0) ack
09:26:54.165528 IP 192.168.205.163.45357 > 10.0.0.101.sunrpc: F
09:26:54.165552 IP 192.168.205.163.45357 > 10.0.0.101.auth: F
```

**Figure 8 : Nmap FIN scan capture**

### Identification tools

#### Operating systems identification tools

The most popular and reliable technique to identify remote operating systems is to generate and compare network stack fingerprints. However in some cases (just like THCrut [thcrut]) banners can be analyzed, as well as combinations of closed and open ports.

- **Nmap OS identification**

Nmap fingerprints are based on the following tests and characteristics :

- Support of TCP options in different orders :
  - WS<sup>1</sup>, MSS<sup>2</sup>, TSV<sup>3</sup>, TSER<sup>4</sup>
  - MSS, WS, TSV, TSER
  - TSV, TSER, WS, MSS
  - TSV, TSER, WS
  - MSS, TSV, TSER, WS
  - MSS, TSV, TSER
- TCP protocol behavior :
  - Initial sequence number (ISN) generation algorithm
  - TCP window size
  - Response to :
    - o SYN, ECN<sup>5</sup>, CWR<sup>6</sup> packet on open port
    - o NULL flags packet on open port
    - o SYN, FIN, PSH, URG packet on open port
    - o ACK packet on closed port
    - o FIN, PSH, URG packet on closed port
- ICMP specificities :
  - ICMP message quoting :
    - o ICMP echo reply : 120 and 150 bytes NULL (0x00) data in ICMP echo request
    - o ICMP port unreachable : 300 bytes data in UDP packet to closed port

---

<sup>1</sup> Window Scale [RFC1323]

<sup>2</sup> Maximum Segment Size [RFC793]

<sup>3</sup> Timestamp Value [RFC1323]

<sup>4</sup> Timestamp Echo Reply [RFC1323]

<sup>5</sup> Explicit Congestion Notification Echo [RFC3168]

<sup>6</sup> Congestion Window Reduced [RFC3168]

## Mitigating Scanners and Crackers

- DF flag support in ICMP echo request

An extract of the packet capture is provided below.

```
34460 > https [SYN] Seq=2057964292 Len=0 MSS=1460
34460 > 3 [SYN] Seq=2057964292 Len=0 MSS=1460
https > 34460 [SYN, ACK] Seq=598710100 Ack=2057964293 win=17520 Len=0 MSS=1460
34460 > https [RST] Seq=2057964293 Len=0
3 > 34460 [RST, ACK] Seq=0 Ack=2057964293 win=0 Len=0
34521 > https [SYN] Seq=2423751763 Len=0 WS=10 MSS=1460 TSV=4294967295 TSER=0
https > 34521 [SYN, ACK] Seq=598768344 Ack=2423751764 win=17520 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
34521 > https [RST] Seq=2423751764 Len=0
34522 > https [SYN] Seq=2423751764 Len=0 MSS=1400 WS=0 TSV=4294967295 TSER=0
https > 34522 [SYN, ACK] Seq=598908726 Ack=2423751765 win=16800 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
34522 > https [RST] Seq=2423751765 Len=0
34523 > https [SYN] Seq=2423751765 Len=0 TSV=4294967295 TSER=0 WS=5 MSS=640
https > 34523 [SYN, ACK] Seq=598970042 Ack=2423751766 win=16640 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
34523 > https [RST] Seq=2423751766 Len=0
34524 > https [SYN] Seq=2423751766 Len=0 TSV=4294967295 TSER=0 WS=10
https > 34524 [SYN, ACK] Seq=599093605 Ack=2423751767 win=16616 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
34524 > https [RST] Seq=2423751767 Len=0
34525 > https [SYN] Seq=2423751767 Len=0 MSS=536 TSV=4294967295 TSER=0 WS=10
https > 34525 [SYN, ACK] Seq=599174006 Ack=2423751768 win=16616 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
34525 > https [RST] Seq=2423751768 Len=0
34526 > https [SYN] Seq=2423751768 Len=0 MSS=265 TSV=4294967295 TSER=0
https > 34526 [SYN, ACK] Seq=599284116 Ack=2423751769 win=16430 Len=0 MSS=1460 TSV=0 TSER=0
34526 > https [RST] Seq=2423751769 Len=0
Echo (ping) request
Echo (ping) reply
Echo (ping) request
Echo (ping) reply
Source port: 34515 Destination port: 30711
```

Figure 9 : Nmap OS scan capture

- **Xprobe2 OS identification**

Xprobe has been created in order to test ICMP stacks and identify OS based on their support of different ICMP packets and options. Since version 0.3 additional tests has been performed in order to makes analysis more accurate; it is now possible to check SNMP sysObjectID, or Netbios banners to make identification more accurate.

Moreover a lot of scan and network mapping modules have been added to Xprobe, making it a more generic scanner than it used to be.

In terms of fingerprinting 9 modules are used.

- 5 ICMP based tests:
  - a. Echo request message quoting and miscellaneous IP parameters (IPID, TOS, TTL)
  - b. Timestamp request support
  - c. Address mask request support
  - d. Port unreachable message quoting
  - e. Information request support
- 2 TCP based tests:
  - f. Connection to an open port. MSS, TSV, Tser, WS TCP options support on an open port, DF option support and miscellaneous TCP parameters, such as TCP window size, sequence numbers ackloedgement etc.

## Mitigating Scanners and Crackers

- g. Packet sent to a closed port, checks for DF option support and IPID strategy.
- 2 application based test:
  - h. SMB NULL session, makes it possible to grab windows information
  - i. SNMP get for sysDescr information

As described in their whitepaper [xprobe-wp] Xprobe is using a fuzzy engine in order to evaluate the type of OS running on a target system. Therefore results are provided with a probability of success that makes the tool more resistant to security measures that would alter the behavior of the target. Below is a sample run of Xprobe.

```
[root@localhost ~]# xprobe2 -p tcp:3:CLOSED -p tcp:443:open -p udp:1:closed -p udp:135:open
-p udp:161:open -p tcp:139:open 10.0.0.106 -v -D 1 -D 2 -D 3 -D 4 -D 5

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 10.0.0.106
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [2] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [3] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [4] fingerprint:icmp_info - ICMP Information request fingerprinting module
[x] [5] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [6] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [7] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [8] fingerprint:smb - SMB fingerprinting module
[x] [9] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 9 modules registered
[+] Initializing scan engine
[+] Running scan engine
[+] All alive tests disabled
[+] Target: 10.0.0.106 is alive. Round-Trip Time: 0.00000 sec
[+] Selected safe Round-Trip Time value is: 10.00000 sec
[+] SMB [Native OS: Windows Server 2003 3790] [Native Lanman: Windows Server 2003 5.2]
[Domain: LAB]
[+] SMB [Called name: LAB6 ] [MAC: 00:01:02:da:83:cf]
[+] Primary guess:
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2003 Server Standard Edition" (Guess
probability: 100%)
[+] Other guesses:
```

## Mitigating Scanners and Crackers

```
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2003 Server Enterprise Edition" (Guess probability: 100%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Workstation" (Guess probability: 95%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Workstation SP1" (Guess probability: 95%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess probability: 95%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess probability: 95%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Workstation SP4" (Guess probability: 93%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Server" (Guess probability: 95%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Server Service Pack 1" (Guess probability: 95%)
[+] Host 10.0.0.106 Running OS: "Microsoft Windows 2000 Server Service Pack 2" (Guess probability: 95%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

**Figure 10 : Xprobe run**

- **THCrut OS scan**

THCrut fingerprints are based on combinations of open and closed ports. On top of it the tool analyzes banners grabbed from different applications to makes it results more accurate. However, a bug in the SNMP implementation often makes the use of this protocol useless.

Tests performed to identify operating systems are the following:

- Tests for ports status
  - a. MS Windows ports : TCP/139, TCP/135, TCP/445, UDP/135
  - b. Unix port : TCP/22
  - c. Cisco port : TCP/2001, TCP/6001
- Tests for banners
  - a. FTP
  - b. SSH
  - c. SMTP
- Test for application data
  - a. Web
  - b. SNMP
  - c. Telnet

These tests are explicitly reported in the verbose mode of THCrut, as show below.

## Mitigating Scanners and Crackers

```
[root@localhost bin]# thcrut discover -v -O 10.0.0.105
thcrut: listening on eth0
thcrut: using source ip 192.168.205.163
Host: 10.0.0.105 0.1.1.5(136448, 00000000.00000010.00010101.00000000) 3 Windows
XP
-----BEGIN THCRUT FINGERPRINT-----
139T=O%135T=O%445T=O%22T=C%2001T=C%6001T=C%
135U=O%
21B=" "
22B=" "
25B="220 lab5 Microsoft ESMTMP MAIL Service, Version: 5\.0\.2195\.1600 ready at
Tue, 23 Oct 2007 14:13:17 \+0200 \r\n"
80W=" Microsoft-IIS/5.0"
161S=" "
23N=" "
-----END THCRUT FINGERPRINT-----
10 packets received by filter, 0 packets dropped by kernel
```

**Figure 11 : THCrut run with fingerprint details**

- **Sinfp OS scan**

Sinfp [sinfp] specificity is to only generate probes to an open port, making it more stealth and difficult to detect. Moreover fingerprints are generated thanks to only 3 tests:

- 2 SYN packets to evaluate multiple TCP stack characteristics : options support (MSS, TSV, TSER, WS), window size, ISN etc.
- 1 ACK packet sent to have the targeted system generate a RST, used to analyze IP and TCP characteristics, such as the DF flag support.

The main characteristic of this tool is that the whole fingerprinting operation is performed with no more than 8 packets, as shown in the capture below.

TCP	59765	>	http	[SYN]	Seq=1071456495	Len=0						
TCP	59766	>	http	[SYN]	Seq=1071456496	Len=0	MSS=1460	TSV=1145389380	TSER=0	WS=1		
TCP	59767	>	http	[SYN, ACK]	Seq=1071456497	Ack=3995447718	win=5840	Len=0				
TCP	http	>	59765	[SYN, ACK]	Seq=821377754	Ack=1071456496	win=16616	Len=0	MSS=1460			
TCP	59765	>	http	[RST]	Seq=1071456496	Len=0						
TCP	http	>	59766	[SYN, ACK]	Seq=821424300	Ack=1071456497	win=17520	Len=0	MSS=1460	WS=0	TSV=0	TSER=0
TCP	59766	>	http	[RST]	Seq=1071456497	Len=0						
TCP	http	>	59767	[RST]	Seq=3995447718	Len=0						

**Figure 12 : sinfp scan capture**

- **OS scanners metrics**

Our analysis doesn't focus on the reliability and success rate of OS scanners, but on their behavior. Therefore the result table will not include results obtained by the four tools analyzed above, but the protocol used, packets sent and received as well as the global packet rate generated by the scan.

**Table 4 : OS scan protocol distribution**

		Nmap	Xprobe2	THCrut	Sinfp
TCP	Bytes	69%	58%	74%	100%
	Packets	86%	62%	83%	100%
UDP	Bytes	9%	27%	15%	0%
	Packets	2%	16%	12%	0%
ICMP	Bytes	22%	15%	11%	0%
	Packets	12%	22%	5%	0%

**Table 5 : OS scan rate based characteristics**

		Nmap	Xprobe2	THCrut	Sinfp
Scan duration (s)		11,5	22	8	<1
<b>Incoming</b>					
Bytes	TCP	4340	747	2521	198
	UDP	0	632	558	0
	ICMP	2120	228	0	0
	<b>Total</b>	<b>6460</b>	<b>1607</b>	<b>3079</b>	<b>198</b>
	<b>Bps</b>	<b>562</b>	<b>73</b>	<b>385</b>	<b>NA<sup>7</sup></b>
Packets	TCP	62	8	19	3
	UDP	0	3	3	0
	ICMP	15	3	0	0
	<b>Total</b>	<b>77</b>	<b>14</b>	<b>22</b>	<b>3</b>
	<b>Pps</b>	<b>7</b>	<b>&lt;1</b>	<b>3</b>	<b>NA</b>
<b>Outgoing</b>					
Bytes	TCP	8170	1119	1708	290
	UDP	1710	295	262	0
	ICMP	1770	240	642	0
	<b>Total</b>	<b>11650</b>	<b>1654</b>	<b>2612</b>	<b>290</b>
	<b>Bps</b>	<b>1013</b>	<b>75</b>	<b>326</b>	<b>NA</b>

<sup>7</sup> These values are not relevant given the low number of packets and data generated

## Mitigating Scanners and Crackers

Packets	TCP	123	12	27	6
	UDP	5	3	3	0
	ICMP	10	4	3	0
	<b>Total</b>	<b>138</b>	<b>19</b>	<b>33</b>	<b>6</b>
	<b>Pps</b>	<b>12</b>	<b>&lt;1</b>	<b>4</b>	<b>NA</b>
<b>Global</b>					
Bps	1575	148	711	NA	
PPS	89	1,5	7	NA	

### Application identification tool

Analysis of tools designed to identify applications is quite complex, as some of them are very generic and used to identify multiple applications, while some others are dedicated to one type of application. Moreover techniques used to identify precisely the nature of a SIP capable device are very different from those used for HTTP or SSH servers.

In order to get a broad view of scanners techniques and characteristics, we will analyze the behavior of three generic scanners, amap [amap] and nmap [nmap], three HTTP dedicated scanners, httpprint [httpprint], hmap [hmap] and nikto [nikto].

- **Generic scanners**

Amap and nmap are very different as the first one relies on signatures while nmap only grabs banner to provide remote application identification.

Amap identification process mainly relies on two mechanisms, triggers and responses. Triggers are data to be sent to specified ports while responses are, obviously, data sent back by the remote application. Most triggers and responses are unrelated. Therefore any data sent to an open port may return a response which is known as specific to an application type, and the remote application will be identified as such. This creates more flexibility as a request doesn't necessarily have to be at the "perfect" format. Moreover it makes it possible to identify applications thanks to error messages.

Below are some examples of triggers and responses as specified in the appdefs.trig and appdefs.resp files [appdefs].

Triggers are defined with the following format :

```
NAME:[COMMON_PORT,[COMMON_PORT,...]][:[IP_PROTOCOL]:0|1]:TRIGGER_STRING
```

Most values are optional and self explanatory. The COMMON\_PORT values are to be used if one specific option (-1) is set, otherwise the trigger will be sent on all ports set for testing. The 0|1 parameters stands for a "harmful" flag and is mandatory.

```
http-get:80,81,82,8000,8080,8081,8888:tcp:0:"GET / HTTP/1.0\r\n\r\n"
http-head:80,81,82,8000,8080,8081,8888:tcp:0:"HEAD / HTTP/1.0\r\n\r\n"
http-proxy-ident:80,81,82,8000,8080,8081,8888:tcp:0:"TRACE HTTP://localhost
HTTP/1.0\r\n\r\n"
http-trace:80,81,82,8000,8080,8081,8888:tcp:0:"TRACE / HTTP/1.0\r\n\r\n"
```

**Figure 13 : amap triggers for http**

Responses are specified with the format:

NAME:[TRIGGER,[TRIGGER,...]]:[IP\_PROTOCOL]:[MIN\_LENGTH,MAX\_LENGTH]:RESPONSE\_REGEX

The TRIGGER is optional and means that this response is to be considered only if it is linked to the specified trigger. RESPONSE\_REGEX is a regular expression using the PERL format (perlre).

```
http-iis::tcp::^HTTP/*\nServer: Microsoft-IIS
http-iis::tcp::^HTTP/*Cookie.*ASPSESSIONID
http-iis::tcp:34:^<h1>Bad Request .Invalid URL.</h1>
http-iplanet::tcp::^HTTP/*Cookie.*iPlanetUserId
http-daap-itunes::tcp::^HTTP/*\nDAAP-Server: iTunes/
http-jrun::tcp::^HTTP/*Cookie.*JSESSIONID
http-jserv::tcp::^HTTP/*Cookie.*JServSessionId
```

**Figure 14 : amap responses for some http servers**

On the opposite nmap uses a trivial banner echoing mechanism to provide version information about targeted applications. Although this technique is easy to fool, it doesn't need updates, which is a great advantage.

Below is an example of a nmap run.

```
[root@localhost ~]# nmap -A -P0 -p21,25,80,135,443,3306 10.0.0.105
Starting Nmap 4.20 ( http://insecure.org ) at 2007-10-24 12:55 CEST
Interesting ports on 10.0.0.105:
PORT      STATE SERVICE      VERSION
21/tcp    open  tcpwrapped
25/tcp    open  smtp         Microsoft ESMT
80/tcp    open  http         Microsoft IIS
135/tcp   open  msrpc        Microsoft Windows
443/tcp   open  ssl/http     Microsoft IIS
3306/tcp  open  mysql        MySQL 5.0.37-community-nt
```

**Figure 15 : nmap run for application identification**

In terms of statistics, the number of TCP sessions established by the scanning source maybe of some interest, in addition to standard metrics such as number of packets, bandwidth etc.

**Table 6 : Generic application scanners metrics**

	amap	Nmap
Scan duration (s)	50	30

## Mitigating Scanners and Crackers

<b>Global traffic</b>					
Packets	Outgoing	1085	22	190	6
	Incoming	808	16	115	4
Total / Rate	<b>Global</b>	<b>1.893</b>	<b>38</b>	<b>305</b>	<b>10</b>
Bytes	Outgoing	92.265	1.845	15.073	502
	Incoming	96.156	1.923	13.219	441
Total / Rate	<b>Global</b>	<b>188.421</b>	<b>3.768</b>	<b>28.292</b>	<b>943</b>
<b>Per application analysis</b>					
Sessions Total / Rate	FTP	28	NA	22	NA
	SMTP	28	NA	3	NA
	HTTP	84	NA	7	NA
	MSRPC	28	NA	3	NA
	HTTPS	56	NA	5	NA
	MYSQL	28	NA	2	NA
Bytes Total / Rate	FTP	12.624	252	10.538	351
	SMTP	25.335	507	1.064	35
	HTTP	21.006	420	1.917	64
	MSRPC	19.647	393	1.558	52
	HTTPS	94.135	1.883	6.722	224
	MYSQL	19.788	396	1.077	35

- **HTTP dedicated scanners**

Dedicated scanners use specific techniques to identify the very nature of an application. On one hand it limits the scope of application, but on the other hand it should provide more accurate results and less “noise”.

The most basic scanner of this family is Nikto. It simply relies on the analysis of the banner received and the behavior of the server responding to two HEAD requests, one of them specifying a total length of 0. The main characteristic of this scanning technique is that it generates no error and a very limited (only 2) number of requests.

On the other hand httpprint and hmap are far more noisy, as they respectively establish 22 and 171 sessions to generate the footprint of the remote web server. Typical tests performed to generate the fingerprint are :

- Support of uncommon but legitimate (according to standards) commands, such as OPTIONS, PUT, TRACE etc.
- Reaction to an erroneous request :
  - Bad command
  - Non HTTP data sent or bad request formatting
  - Bad protocol name or version

## Mitigating Scanners and Crackers

- Behavior in extreme conditions
  - Large URI
  - Repetitive schema in URL

Below are a few examples of tests performed by httpprint and amap against an IIS 5.0 and an Apache 2 server. Below the figures shows the results of two hmap runs against those two different servers.

**Table 7 : HTTP fingerprint samples**

Request	IIS 5.0	Apache 2.0.52
<b>Httpprint</b>		
GET / JUNK/1.0	400 Bad Request	200 Ok
get / http/1.0	400 Bad Request	501 Not implemented
POST / HTTP/1.0	405 Method Not Allowed	200 Ok
GET /cgi-bin/	404 Not Found	403 Forbidden
<b>Hmap</b>		
MKCOL / HTTP/1.0	403 Forbidden	405 Method Not Allowed
PROPFIND / HTTP/1.0	411 Length Required	405 Method Not Allowed
GET	400 Bad Request	<i>Non HTTP response</i>
GET / HTTP/999.99	200 Ok	400 Bad Request

```
[root@localhost hmap]# python hmap.py http://10.0.0.101:80
gathering data from: http://10.0.0.101:80

                                matches : mismatches : unknowns
Apache/2.0.40 (Red Hat 8.0)          110 :          4 :    9
Apache/2.0.44 (Win32)               109 :          5 :    9
IBM_HTTP_Server/2.0.42 (Win32)     108 :          6 :    9
Apache/1.3.9 (Win32)                107 :          8 :    8
Apache/1.3.12 (Win32)              107 :          8 :    8
```

**Figure 16 : hmap run against Apache 2**

```
[root@localhost hmap]# python hmap.py http://10.0.0.105:80
gathering data from: http://10.0.0.105:80

                                matches : mismatches : unknowns
Microsoft-IIS/5.0 (Win32)          102 :         14 :    7
Apache/1.3.23 (RedHat Linux 7.3)   53 :         62 :    8
```

## Mitigating Scanners and Crackers

Apache/1.3.27 (Red Hat 8.0)	53 : 62 : 8
Apache/2.0.44 (Win32)	51 : 64 : 8
Apache/1.3.26 (Solaris 8)	51 : 64 : 8

**Figure 17 : hmap run against IIS 5.0**

However, the number and rate of requests are not the only metrics of interest as they may not be so uncommon. On the other hand the number of errors (or uncommon response codes) may be a valuable piece of information.

**Table 8 : HTTP identification scanners metrics**

		Nikto		httpprint		Hmap	
		IIS 5.0	Apache 2	IIS 5.0	Apache 2	IIS 5.0	Apache 2
Scan duration (s)		0,007	0,004	10	0,040	24	59
HTTP Sessions		2	2	22	22	171	199
<b>Responses</b>							
<b>OK</b>		<b>2</b>	<b>2</b>	<b>9</b>	<b>11</b>	<b>58</b>	<b>62</b>
<b>Errors</b>	3xx	0	0	0	0	0	0
	4xx	0	0	9	8	89	81
	5xx	0	0	1	2	18	12
	Non-http	0	0	3	1	6	44
	<b>TOTAL</b>	<b>0</b>	<b>0</b>	<b>13</b>	<b>11</b>	<b>113</b>	<b>137</b>
<b>Proportion</b>	<b>0%</b>	<b>0%</b>	<b>59%</b>	<b>50%</b>	<b>66%</b>	<b>69%</b>	

These results logically vary from on system to another, as this is the very mechanism which the fingerprinting technique relies on. However, results clearly show that globally the scale remain the same in terms of error proportion.

### Cracking tools

The term cracking tools includes many different kind of tools. Most common ones are vulnerability scanners. Used to identify systems weaknesses they are widely used by malicious individuals, automated worms as well as security administrators.

However, it is also interesting to focus on two other categories of tools : mass generators and attack tools. Tools that belong to those two categories are real malicious tools that will actively compromise remote systems' security.

Therefore being able to block these tools is an essential step to maintain the security of any IT infrastructure.

#### Vulnerability scanners

Vulnerability scanners are designed to perform several tests in a more or less subtle way. In some cases, tools will be able to identify the remote system first and then launch only appropriate tests. In other cases, the user will have to specify the type of target to analyze. Last, some tools will blindly launch any kind of attack they know and check the exposure of the target.

On top of these operational differences each and every vulnerability scanner uses different techniques to perform their tests. Some will compare version numbers with a database of known vulnerabilities, some other will evaluate response to specific stimuli, other will launch real attacks. Therefore statistics and analysis cannot rely on specific test techniques but on more generic characteristics.

In order to illustrate those differences we analyzed 3 different scanners, Nessus [nessus], nikto [nikto] and NTOinsight [ntotools].

- **Operation modes**

#### Nessus

Nessus is probably the most famous security scanner freely available. Relying on a real client-server architecture it has a professional design from the very start, about ten years ago.

Nessus is provided with "plugins" that makes it totally autonomous. However it can get information from external tools such as nmap or amap. During the scanning process Nessus launches a few port scan probes then focuses on the exploitation of miscellaneous possible vulnerabilities.

These tests can be classified in 6 categories

1. Identification and information grabbing based on public information and methods, such as banner grabbing, robot.txt analysis etc.
2. Identification of supported commands, such as TRACE, PROPFIND etc.
3. Dictionary attack to discover common directories, such as /backup, /admin, /include etc.
4. Research of known vulnerable scripts and applications such as awstat.pl, smb2www.pl, phorum.php etc.

## Mitigating Scanners and Crackers

### 5. Blind exploit attempts of known vulnerabilities such as :

- Directory crossing :  
`/cgi-bin/view_source?../../../../../../../../../../../../etc/passwd`
- Cross-site scripting :  
`/scripts/user.cgi?url=">%3Cscript%3Ealert("gossamer_links_url_xss.nasl")%3B%3C%2Fscript%3E&from=add`
- Global variables tampering  
`/calendar.php?serverPath=/etc/passwd%00`
- SQL injections :  
`/cgi-bin/SPT--ForumTopics.php?forumid=-9%20UNION%20SELECT%20null%2cnull%2cnull%2c1194622963%2c4%2c5`

### 6. Targeted exploits attempts :

- Research of hidden or old files :  
`GET /cgi-bin/badstore.cgi.old HTTP/1.1\r\n`
- SQL injections on GET parameters in URL :  
`GET /cgi-bin/badstore.cgi?-=&action='+OR+1=1# HTTP/1.1\r\n`

One the main characteristics of Nessus HTTP scans is that it heavily uses HTTP 1.1 capability to pipeline multiple requests in a single TCP sessions. During the scan Nessus pipes 101 requests per TCP sessions. This saves resources and makes scan faster.

## Nikto

Nikto is a far more basic tool wrote in PERL that performs simple operations based on the following mechanisms.

1. Grab server banner and define a generic category for the server, based on strings returned in the response to a HEAD request. Categories and matching strings are defined in the `server.db` file.

```
#####  
# Server categories: "category", "match-string"  
#####  
"abyss", "abyss"  
"alchemyeye", "Alchemy Eye"  
"apache", "apache"  
"apache", "apache-coyote"  
"apache", "infrastructure"  
"apache", "jakarta"  
"apache", "tomcat"  
"apache", "IBM_HTTP_SERVER"  
"cern", "cern"
```

**Figure 18 : extract of server.db**

## Mitigating Scanners and Crackers

2. Evaluate web servers exposure to known vulnerabilities by comparing version information and vulnerability database. This operation is static and each and every vulnerability is hard coded in the file `server_msg.db`. The format is simple : `<server_string_regexp> <vulnerability_description>`. Below is a sample entry of the file.

```
"Apache\/(1\.2\.([2-9].*|1[0-9])|1\.3\.([0-1].*|2[0-4]))","Apache 1.x up 1.2.34 are vulnerable to a remote DoS and possible code execution. CAN-2002-0392."
```

**Figure 19 : server-msg.db sample entry**

3. Test for potential vulnerabilities in scripts. As of today all tests are blindly launched to the target. However, the format of the file `scan_database.db` implies that there is (or was) a plan to make tests more accurate. Indeed each line of the file has the following format :

```
<category> , <URL> , <response_string> , <http_command> , <vulnerability_description>
```

In the current file, the category field is always set at "generic". All those tests are hard coded and the only potential variation is the manual setting for CGI directories that can be specified via the command line option `-Cgidirs`.

4. Default password attacks based on blind and targeted attempts, based on the authentication realm required by the server. These tests are defined in the `realms.db` file.

Additionally Nikto implements evasion techniques, based on URL encoding and makes it possible to create plugins. However, only a few of them have been developed and are part of the original distribution.

### **NTOInsight**

NTOInsight is a little bit different as it performs web site analysis on top of a Nikto-based vulnerability scan. This web analysis provides information for further exploitation attempts based on application specific attack points. These attack points are web pages serving the following kind of content:

- Forms;
- Query strings (parameters embedded in the URL)
- Authentication page;
- Cookies;
- Scripts;
- Hidden fields;
- Applets;
- Email.

## Mitigating Scanners and Crackers

For each and every attack point NTOInsight provides a full report detailing parameters name, request targets, cookies name and values etc. Below is the detail of one attack point detected by NTOinsight.

**URL:** http://10.0.0.207:80/cgi-bin/badstore.cgi?action=myaccount  
**Response Code:** 200 **Page Signature:** 200:FBFAAA631AE6A0838C5B67A30BC130503FB2C358  
**Resources:**

**Forms:**

Name	Method	Fields	Hidden Fields	Action
search	get	3	1	/cgi-bin/badstore.cgi?action=myaccount
<b>Field Name</b>				<b>Field Type</b>
searchquery				text
action				hidden
				image

Name	Method	Fields	Hidden Fields	Action
_form1	POST	2	0	/cgi-bin/badstore.cgi?action=moduser
<b>Field Name</b>				<b>Field Type</b>
email				text
DoMods				submit

**Cookies sent in request:**

Name	Expires	Domain	Path	SSL Only	Value
SSOid	End of Session	/	No		OmQ0MWQ4Y2Q5OGYwMGlyMDRIOTgwMDk5OGVjZjg0 MjdlOj0%3D%0A

**Query Strings:**

<b>Query String:</b> http://10.0.0.207:80/cgi-bin/badstore.cgi?action=myaccount	
<b>Name:</b> action	
<b>Value:</b> myaccount	

**Figure 20 : Attack point detail**

- **Attack metrics**

Once again the number of request and the rate at which they are generated are not necessarily relevant as they may not be particularly uncommon, especially on popular sites and compared to traffic generated by meta-proxies.

For this reason we will focus on application level information such as the number of request and the success rate.

**Table 9 : Vulnerability scanners metrics**

	Nessus	Nikto	NTOInsight
Scan duration (s)	88	300	95
HTTP Requests	3858	16729	2387
Requests/s	43	55	25

## Mitigating Scanners and Crackers

Responses				
OK		83	64	26
Errors	3xx	7	1	0
	4xx	3658	16652	2358
	5xx	1	6	3
	Non-http	109	6	0
	<b>TOTAL</b>	<b>3775</b>	<b>16665</b>	<b>2361</b>
	<b>Proportion</b>	<b>98%</b>	<b>99%</b>	<b>99%</b>

Whatever is the technique used, and whenever deep investigation is performed on the web site, ratios are all quite the same and very unlikely to be commonly found in “normal” users behaviors.

### Mass generators and attack tools

- **Password crackers**

Most interesting and common mass generators are password crackers, used to perform thousands of tests against authentication mechanisms and “guess” login credentials. Guesses are usually done via two techniques : dictionary and brute force attacks.

Dictionary attacks rely on the use of big word files (commonly over one million words) to test as login and passwords. According to the context these files can be generic, usually for the password, or specially crafted to fit username formats. Attacks to test known default login / password pairs are obviously based on this technique. The main advantage of such technique is that it is relatively quick (a few hours are enough to test for hundreds of thousand combinations) and can be efficient if users don't apply basic security rules in the choice of their passwords. Moreover a smart attacker will try to narrow the range of possible passwords by using language specific dictionaries and including context related words, such as the company name, personal information etc. On the other hand the range of test is not exhaustive and complex passwords will remain undiscovered.

Brute force attacks will try any possible combination of a set of character. Therefore the only choice to do is that of a range of possible characters that may be in the password. Of course the larger is the character set the longer will be the cracking operation. Eventually most of inline brute force operations are limited to a set of 62 characters, which are 10 numbers and 2x26 letters (lower and upper case). In such schema, the brute force attack is not much more efficient than dictionary attacks but far slower. That's the main reason why there are not much of these tools for inline cracking.

Whatever technique is used, the most obvious characteristic of password cracking operations is that the ratio of authentication failures will be close to 99%. Therefore the interesting metric is immediately found. Other criteria, such as packet rates or bandwidth will, once again, not appear as really noticeable, especially on “popular” applications during rush hours. As an example, a huge amount of authentication request at 09:00 am on a corporate mail server (POP or IMAP) is usually not to be considered as suspicious.

## Mitigating Scanners and Crackers

- **Attack tools**

Attack tools are the ultimate automation step, that makes hacking faster and easier. The most popular attack tool is Metasploit [metasploit]. This tool will launch a an attack that has been specified and automate the exploitation process so that the “user” doesn’t have to perform any complex operations, such as creating a new user, uploading a malicious software, or even loading a graphical server in the compromised system memory... leaving almost no trace of the intrusion. However, Metasploit and the likes just launch a single attack and cannot really be considered as scanners or crackers.

On the other hand the exploitation of some potential vulnerability can be quite long and require many tries in order to exactly match the target system specificities. This is especially the case of SQL injections, where quoting, commenting and even performing simple operations such as JOINT or UNION is always system specific. In this case tools like SQLiX [sqlix] will perform multiple tries automatically to find if the system is vulnerable and the way the vulnerability can be exploited.

However, if the checks are automated, the user has to specify which field he wants to test. In this situation, the investigation performed by scanners such as NTOinsight, analyzed before, is very valuable as it provides all the potential vulnerable fields.

Below is a sample run of SQLiX.

```
[root@localhost SQLiX_v1.0]# ./SQLiX.pl -v=2 -url="http://10.0.0.205/cgi-bin/badstore.cgi?action=login&passwd=a&email=a" -exploit -all
=====
-- SQLiX --
© Copyright 2006 Cedric COCHIN, All Rights Reserved.
=====

Analysing URL [http://10.0.0.205/cgi-bin/badstore.cgi?action=login&passwd=a&email=a]
http://10.0.0.205/cgi-bin/badstore.cgi?action=login&passwd=a&email=a
[+] working on action
    [+] Method: MS-SQL error message
    [+] Method: SQL error message
    [+] Method: MySQL comment injection
    [+] Method: SQL Blind Statement Injection
    [+] Method: SQL Blind String Injection
[+] working on passwd
    [+] Method: MS-SQL error message
    [+] Method: SQL error message
    [+] Method: MySQL comment injection
        [ERROR] Parameter doesn't impact content
    [+] Method: SQL Blind Statement Injection
        [ERROR] Parameter doesn't impact content
    [+] Method: SQL Blind String Injection
        [ERROR] Parameter doesn't impact content
[+] working on email
    [+] Method: MS-SQL error message
    [+] Method: SQL error message
        [FOUND] Match found INPUT:['] - "You have an error in
your SQL syntax"
        [INFO] Error with quote
        [INFO] Database identified: MySQL Server
```

## Mitigating Scanners and Crackers

```
[INFO] Current function: version()
[INFO] length: 14
        4.1.7-standard
[FOUND] SQL error message

RESULTS:
The variable [email] from [http://10.0.0.205/cgi-bin/badstore.cgi?action=login&passwd=a&email=a] is vulnerable to SQL Injection
[Error message (') - MySQL].
```

**Figure 21 : SQLiX run**

One of the particularities of SQLiX is its native capability to obfuscate its attacks packets. An example of an obfuscated attack on the email field is provided below.

```
GET /cgi-bin/badstore.cgi?action=login&passwd=a&email=1'%2B1%20regex%20IF((ascii(substring(version()%2C12%2C1))%3E%3E(4)%261)%2Cchar(42)%2C1)%20AND%201%3D1%2B'
HTTP/1.1\r\n
```

**Figure 22 : SQL injection obfuscated by SQLiX**

This attack aims at retrieving the version number of the database, through a field vulnerable to the basic `OR 1` SQL injection. In this specific request, SQLiX is going to check the value of the 5<sup>th</sup> bit of the 12<sup>th</sup> character of the version string. Most of special characters, like commas, + or > signs are encoded in hexadecimal. This test is performed by the following functions :

```
Function(1) : ascii(substring(version(),12,1))>>(4)&1
```

- `version()` returns a string with the version number
- `substring(version(),12,1)` return the 12th character
- `ascii(substring(version(),12,1))` return the numeric value of the character
- `ascii(substring(version(),12,1))>>(4)` right shifts 4 bits
- `ascii(substring(version(),12,1))>>(4)&1` checks if the rightmost bit equals 1

```
Function (2) : IF(ascii(substring(version(),12,1))>>(4)&1,*,1)
```

A IF control is operated (Function 2), it will return `*` if the result of Equation 1 is TRUE (1) and `1` if the result is FALSE.

Then a pattern matching is performed between the string 1 and the result of the IF condition, via the regexp function (Function 3).

```
Function (3) : 1 regexp IF(ascii(substring(version(),12,1))>>(4)&1,*,1)
```

Therefore there are two possibilities :

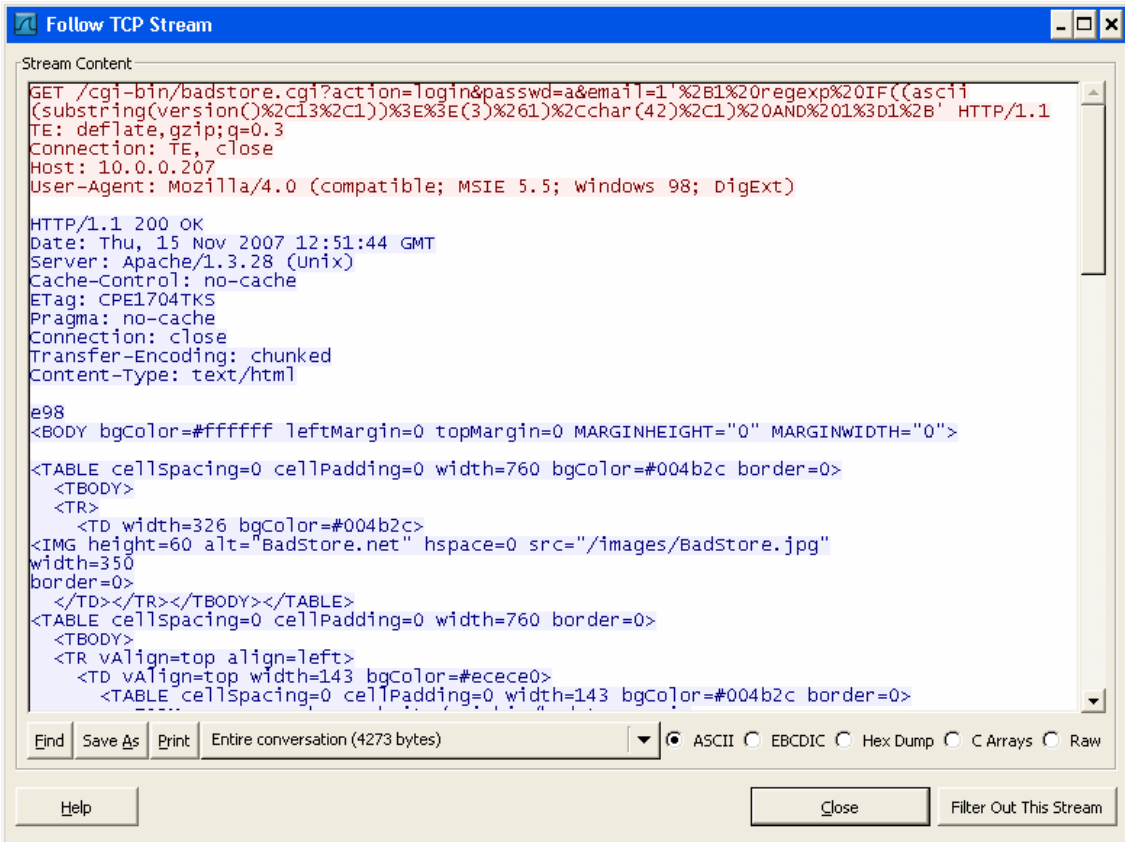
1. `1 regexp *` : generates an SQL error

## Mitigating Scanners and Crackers

2. 1 regexp 1 : returns 1

Eventually the injection ends in a simple form if the result of Equation 1 is TRUE: `email=1' 1 AND 1'`, While a result of FALSE will generate an SQL error and serve a different page.

In terms of metrics the main problem is that there is no major difference between a successful check and an error, as all of them will return a HTTP 200 OK code but with different contents, as shown in Figures below.



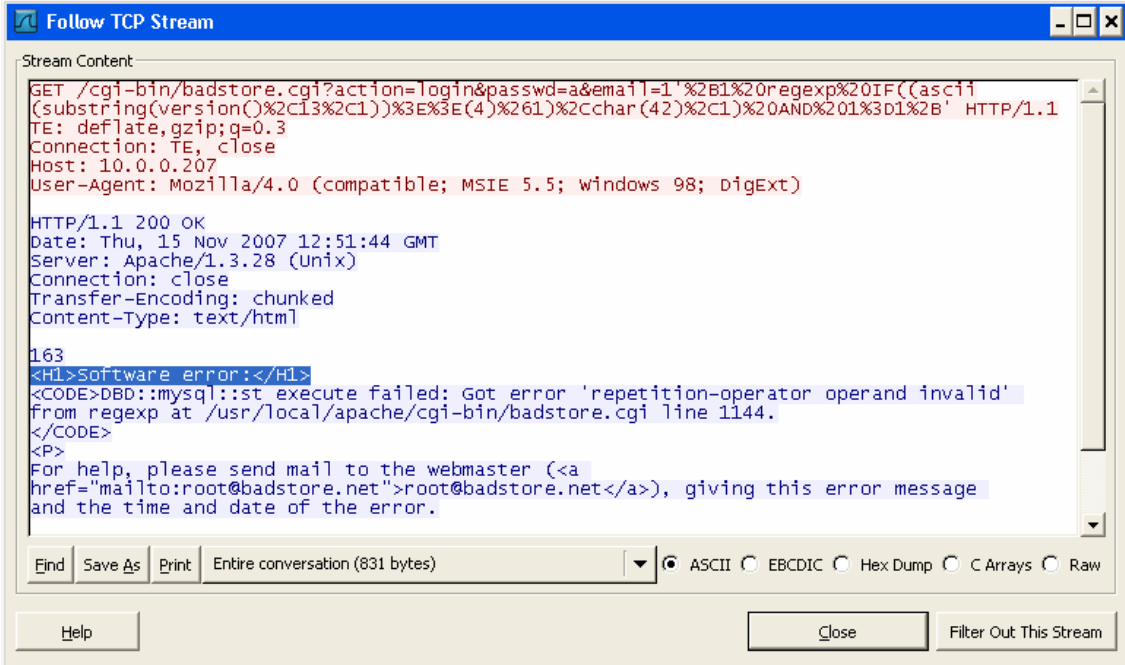
```
Stream Content
GET /cgi-bin/badstore.cgi?action=login&passwd=a&email=1'%2B1%20regexp%20IF((ascii
(substring(version()%2C13%2C1))%3E%3E(3)%261)%2Cchar(42)%2C1)%20AND%201%3D1%2B' HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: 10.0.0.207
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; windows 98; DigExt)

HTTP/1.1 200 OK
Date: Thu, 15 Nov 2007 12:51:44 GMT
Server: Apache/1.3.28 (Unix)
Cache-Control: no-cache
ETag: CPE1704TKS
Pragma: no-cache
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

e98
<BODY bgColor=#ffffff leftMargin=0 topMargin=0 MARGINHEIGHT="0" MARGINWIDTH="0">
<TABLE cellSpacing=0 cellPadding=0 width=760 bgColor=#004b2c border=0>
  <TBODY>
    <TR>
      <TD width=326 bgColor=#004b2c>
        <IMG height=60 alt="BadStore.net" hspace=0 src="/images/BadStore.jpg"
width=350
border=0>
      </TD></TR></TBODY></TABLE>
<TABLE cellSpacing=0 cellPadding=0 width=760 border=0>
  <TBODY>
    <TR valign=top align=left>
      <TD valign=top width=143 bgColor=#ecec0>
        <TABLE cellSpacing=0 cellPadding=0 width=143 bgColor=#004b2c border=0>
```

Figure 23 : SQL injection response with valid SQL

## Mitigating Scanners and Crackers



The screenshot shows a window titled "Follow TCP Stream" with a text area containing the following content:

```
Stream Content
GET /cgi-bin/badstore.cgi?action=logIn&passwd=a&email=1'%2B1%20regexp%20IF((ascii
(substrstring(version()%2c13%2c1))%3E%3E(4)%261)%2cchar(42)%2c1)%20AND%201%3D1%2B' HTTP/1.1
TE: deflate, gzip;q=0.3
Connection: TE, close
Host: 10.0.0.207
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; windows 98; DigExt)

HTTP/1.1 200 OK
Date: Thu, 15 Nov 2007 12:51:44 GMT
Server: Apache/1.3.28 (Unix)
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

163
<H1>Software error:</H1>
<CODE>DBD::mysql::st execute failed: Got error 'repetition-operator operand invalid'
from regexp at /usr/local/apache/cgi-bin/badstore.cgi line 1144.
</CODE>
<P>
For help, please send mail to the webmaster (<a
href="mailto:root@badstore.net">root@badstore.net</a>), giving this error message
and the time and date of the error.
```

At the bottom of the window, there are buttons for "Find", "Save As", "Print", and "Entire conversation (831 bytes)". To the right of these buttons are radio buttons for "ASCII" (selected), "EBCDIC", "Hex Dump", "C Arrays", and "Raw". At the very bottom, there are buttons for "Help", "Close", and "Filter Out This Stream".

Figure 24 : SQL injection response with invalid SQL

## Mitigating the threat

### The challenge of mitigation

#### Axis of research

Blocking mass scanning and cracking activities will noticeably increase the security level of an IT infrastructure as it will become capable of defend itself against most automated tools, would they be autonomous, like worms, or used to quickly gather information and perform (in)security checks, like would do a malicious user.

Designing a system able to detect and block such threats makes it necessary to identify common criteria. From this point it should be possible to analyze potential anomalies and create an engine, based on network metrics "behavior".

Another challenge is the capability to evaluate the efficiency of such engine, and to understand its limitations. 100% security is illusory, but weaknesses in the security are acceptable as long as they are identified. Therefore it becomes mandatory to understand the limits of the system we would design.

#### The need for new technologies

One of the most obvious characteristics of scanners and crackers is their flexibility. As an example the nmap "ping", designed to identify the status of networked systems, can be performed via 3 mechanisms: ICMP, ICMP + TCP, TCP only. Nessus identifiers, such as the username used for authentications or the User-Agent of the internal web client, can be changed by the user. In the case of non-public malicious software, such as worms, there are even more variants and one can bet that scan packets generated by two different worms or bots will not have any common content.

Therefore it is clear that prevention technologies based on simple pattern matching and accurate threat recognition cannot be efficient on a large scale, for at least two reasons. The first one is that known tools, bots and worms (and their main variants), would already involve thousands of "signatures" for their accurate detection. On top of it there is no obsolete signature as a port scanner or a password cracker remains efficient as long as the supporting protocols (IPv4 suite today) are used. This huge amount of detection patterns will necessarily impact performances.

The second reason is that one of the characteristics of these threats the rapidity of their propagation. The more quickly they propagate, the more system will be compromised before the tool or code gets analyzed and a signature is found. Malicious code propagation is obviously faster than signature design and deployment. Therefore any signature-based protection will likely miss the outbreak of the malicious code propagation.

#### Common prevention issues

Along with the detection capabilities, the accuracy of identification and characterization of the threat is a critical point. First it will make it possible to block all malicious packets and ensure that the threat is completely mitigated. Second it will ensure that legitimate traffic remains unaffected. Indeed, one of the most negative drawbacks of active security

## Mitigating Scanners and Crackers

devices is their “capability” to interrupt production traffic while under attack. This misbehavior is sometimes voluntarily caused by malicious individuals whose goal is to disrupt network operations.

Another issue is the never ending debate about the necessity to completely blacklist a system that appears to be the source of a scanning or propagation activity. Blacklisting a user workstation can be realistic, as long as the system is physically accessible by a security engineer who will fix the issue locally. Of course this procedure is not applicable to remote offices where no IT competency is available. Therefore the blacklisting will result in a loss of productivity for the owner of the workstation.

An alternative case, which is quite similar although more critical in terms of impact, is that of a server being compromised. In such case blacklisted server will block any operations until the issue is fixed. It clearly means that the loss of productivity will not impact a single user as it was the case before, but any user who needs data or applications served by the compromised system. On top of it,, if the compromised system appears to be a publicly accessible one, such as a web server, the blacklisting will result in a widely noticeable disruption of service and will quite likely impact the image of the company.

These issues of accuracy and blocking scope are to be considered in the design of a security mechanism intended to block malicious activity. Otherwise the mitigation affects may be worst than those of the attack.

### **Detection metrics**

#### **Global networking activities**

Networking activity is usually measured thanks to two criteria, packet rate and bandwidth. Scanners and crackers are not tools designed to generate bandwidth, however bandwidth consumption maybe a side effect of some scanning activities even if not to be considered as a metric directly related to the scan activity.

**Table 10 : Global networking activities**

Operation	PPS	Bandwidth (Mbps)
-----------	-----	------------------

## Mitigating Scanners and Crackers

<b>L4 Scans</b>		
Nmap Connect	1667	0,510
Nmap SYN	1673	0,789
Nmap FIN	839	0,383
Nmap XMAS	1662	0,758
Hping SYN	501	0,229
Hping FIN	224	0,102
Hping XMAS	224	0,102
<b>OS Scans</b>		
Nmap	89	0,012
Xprobe2	1,5	0,001
THCrut	7	0,006
<b>Application Scans</b>		
Amap	38	0,025
Nmap	10	0,008
HTTPrint	20	0,038
Hmap	161	0,876
<b>Vulnerability scans</b>		
Nikto	555	0,518
Nessus	182	0,423
NTOinsight	253	0,273
<b>Attack Tools</b>		
SQLiX	24	0,255

Unfortunately it clearly comes out that these metrics will not make it possible to detect all scanners and cracker activities. Moreover, even the highest numbers observed (around 0,8 Mbps and 1600 pps) are not high enough to raise any alert or to clearly identify a suspicious source. With these metrics false-positive risk is definitely unacceptable.

### Detecting errors

Most of scanners' operations are based on blind testing of the remote system. Therefore it is obvious that they will generate errors, would it be at the networking or application level. This approach seems even more appropriate as some tools rely on errors, voluntarily generated to evaluate the behavior of the remote system.

Then, for any protocol that involves error generation, it is interesting to observe the error rate of malicious activity.

**Table 11 : Erroneous responses**

## Mitigating Scanners and Crackers

Operation	Error type	Error rate
<b>L3 Scans</b>		
Fping ECHO	ICMP Host Unreachable	77%
THCrut ECHO	ICMP Host Unreachable	40%
THCrut MASK	ICMP Host Unreachable	60%
Nmap ECHO	ICMP Host Unreachable	33%
Nmap MASK	ICMP Host Unreachable	53%
Nmap TSTAMP	ICMP Host Unreachable	33%
<b>L4 Scans</b>		
Nmap Connect	RST	99%
Nmap SYN	RST	99%
Nmap FIN	RST	100%
Nmap XMAS	RST	100%
Hping SYN	RST	99%
Hping FIN	RST	100%
Hping XMAS	RST	100%
<b>Application Scans</b>		
Nikto	HTTP Response Code > 400	0%
HTTPPrint	HTTP Response Code > 400	55%
Hmap	HTTP Response Code > 400	67%
<b>Vulnerability scans</b>		
Nikto	HTTP Response Code > 400	99%
Nessus	HTTP Response Code > 400	98%
NTOinsight	HTTP Response Code > 400	99%
<b>Attack Tools</b>		
SQLiX	HTTP Response Code > 400	0%
Hydra	HTTP Authentication Failre	> 99%

Focusing on errors provides a better view of potential malicious activity. For port scans, vulnerability scans and password crackers error ratios are around 99%, which doesn't leave a doubt about the nature of the operation. Application scans also have good ratios as fingerprinting tools get about 60% of error responses, far above average of a normal behavior. Even L3 scans get "good" results as it is not common to get 33% to 77% of ICMP error messages.

However, two categories of tools remain undetectable : L2 scanners, as ARP doesn't generate errors when a physical address is requested for a non-existing IP address and

## Mitigating Scanners and Crackers

OS scans, that rely on various test, and therefore may generate different errors. In this very case analyzing errors would require first to analyze each and every tool and create a set of relevant errors. This would be similar to working with signatures, with the limitations we described above.

### **Tuning error statistics**

Two of the “appropriate” categories of malicious activity identified above may not be able to generate the corresponding error message: L3 and L4 scanners, as they may be blocked by firewalls, simply dropping the request to the destination. In such specific case one security mechanism will prevent another one from being efficient.

One solution would be to evaluate the global communication and not only analyze packets sent from the target to the source.

At ICMP level three level of communication can be defined :

1. ICMP REQUEST / No Response
2. ICMP REQUEST / ICMP ERROR
3. ICMP REQUEST / ICMP REPLY

For TCP 5 types of communication can be identified

1. SYN / No Response
2. SYN / RST
3. SYN / SYN-ACK / ACK / No Response
4. SYN / SYN-ACK / ACK / [Data] / RST
5. SYN / SYN-ACK / ACK / [Data] / FIN / FIN-ACK / ACK

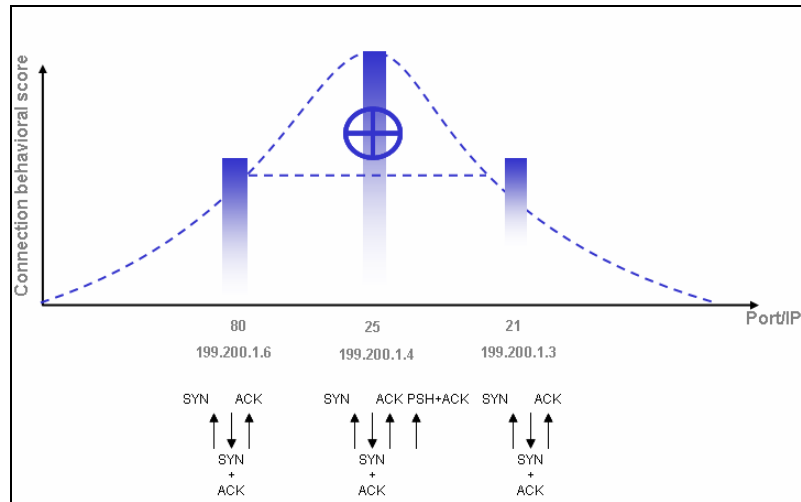
And UDP will support 3 types of communication

1. Data / No Response
2. Data / Response
3. Data / ICMP ERROR

If a weight is given to each category, accordingly to the probability of being a “normal” behavior (lower = suspicious, higher = legitimate), it becomes possible to create a model for each source address.

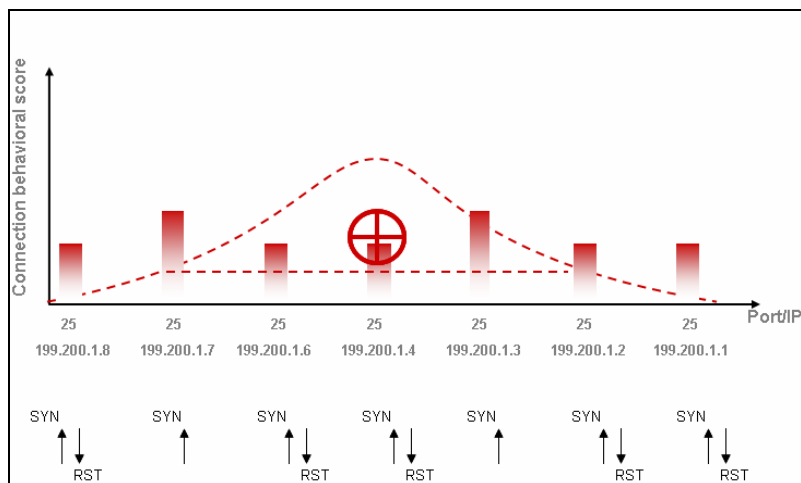
In the case of legitimate activity most communication types have an heavy weight, and the communications can be visualized as in the graph below :

## Mitigating Scanners and Crackers



**Figure 25 : Legitimate user connection profile**

On the opposite a malicious source which intends to scan networks and systems, looking for available applications will have the following model, which applies to scanners as well as worms attempts to propagate.



**Figure 26 : Malicious source connection profile**

## Characterization and blocking

### Common methods and limitations

- **Basics of source blacklisting**

Once a source is identified the most obvious way to prevent it from propagating malwares or scanning a network is to block any traffic generated by the source. As it has been discussed before this method makes intrusion characterization trivial but has many drawbacks as it is prone to block legitimate traffic and to be leveraged by malicious users to generate denial of service. However, in a few cases it may be interesting to consider using this method.

First the risk of production loss caused by the blocking of a company's resource is limited to systems that belong to the company network. It means that blocking external systems as source of potential intrusion should not have the production impact of the same operation performed on "internal" systems.

However, spoofing of source address is still something possible. A malicious user could then simulate a scan initiated by a fake source that would be quickly blacklisted. This remains possible as long as malicious operations do not require a complete connection. Therefore it is very unlikely that a source would be spoofed and the blocking of a specific source may be considered as an appropriate solution. This is the case, in particular for cracking tools.

- **Improving blacklisting**

But cracking tools may still be launched from a remote system that would have been compromised, or behind a proxy. In this case a big part of legitimate traffic will be dropped and this may not be acceptable, even in the case of traffic coming from the outside. There are two solutions that can be applied to reduce the side effects of this technique.

The first one is to limit the scope of the blocking. Instead of blocking any traffic coming from a source it is realistic to block only the traffic coming from a specific source and targeting a single destination port. This technique will prevent a global blocking of all traffic initiated by the source host, which could be compromised. In such case a mail server, hacked and used to scan for Windows RPC vulnerabilities (port TCP/135), will still be able to send mails while all the RPC traffic it initiates will be blocked. In the same To a certain extent "targeted blacklisting" will also be efficient against password cracking attempts.

The second improvement is to define increasing blocking time, with a pre-defined maximum. Once the current aging time is expired detection engine starts again. If an identical source IP / destination port pair is found it will get blocked for a longer time and so on. This technique should prevent much false positives and reduce the impact of blocking legitimate traffic to an acceptable minimum. Indeed, it may be possible that for some reasons, a legitimate automated tool generates a high error rate. However, it is quite unlikely that this problem will recursively occur for ages. On the other hand worms don't get "fixed". Depending on their implementation they will either abort or keep on trying. In the first case the first short period blocking will efficiently stop the worm

propagation. In the second case, long term blocking is a solution to slow down propagation in such a manner that it will take ages only to scan a /28 network.

### Footprinting the threat

- **The idea of footprinting**

However improved blacklisting can be, it will only show efficiency against worms propagation, and especially from outside the protected network. It is then necessary to investigate other identification methods that will make it possible to better qualify the offending traffic and activate the blocking mechanism against this specific traffic.

Ideally the prevention system should both behave as a signature-like mechanism, providing accurate identification of a threat, and perform dynamic analysis of potential threats to create filters on the fly. This would make it possible to identify any kind of malicious traffic without the need for upstream analysis of each and every existing tool or malicious piece of code, and to limit the blocking scope to that of suspicious traffic, leaving legitimate data without disruption of any kind.

This dynamic identification of a potential malware can only be performed thanks to generic characteristics and has to be totally independent from the content and even the very nature of the threat; the purpose being to accurately block scans, cracks or even worms without any prior knowledge about the objective of the malicious operation, the targeted application or the tested vulnerabilities.

The idea of footprinting is to create an identification pattern based on observed traffic. While the purpose of fingerprinting is to find a match between a pre-defined pattern and an occurring phenomenon, footprinting creates the pattern by analyzing the phenomenon. Obviously this operation can only be efficient if the phenomenon itself has been properly identified and isolated, and can only rely on an accurate detection mechanism, such as the error-based technique described earlier.

- **Footprints characteristics**

As previously stated, characteristics used to find footprints must be common to all potential threats. Therefore it is logical to start from the bottom of the network stack which is the common basement for all communications.

However, starting at Layer 2 with Ethernet protocol doesn't provide much information. Indeed, the Ethernet frame does not contain much information that could specifically be used to discriminate a specific threat. Moreover, source MAC addresses are by no mean useful for any traffic originated from outside the IP network where the data are gathered.

IP is more interesting as many fields' values may vary from one packet to another, according to the fact that they are generated by the generic IP stack of the system or hand crafted by a packet generation tool like hping or by the malicious code itself. These fields are given in the table below.

**Table 12 : Layer 3 footprint characteristics**

Field	Size (b)	Characterization use
Differentiated Services	4	Usually unused

## Mitigating Scanners and Crackers

Total Length	16	Identical content identification
IPID	16	Should be random
TTL	8	Should vary from one source to another
Options	< 40 B	Usually unused

Although we get some interesting pieces information, a set of five potential characteristics is not enough to accurately characterize a threat and it is necessary to get additional information from the transport layer.

At layer 4 we encounter two extremes, with TCP being particularly rich on one hand, UDP and ICMP carrying almost no information that could be exploited for footprinting on the other hand.

**Table 13 : Layer 4 footprint characteristics**

Field	Size (B)	Characterization use
<b>TCP</b>		
Source port	2	Random and changing
Sequence number	4	Random and changing
TCP Window	4	OS specific
TCP options	-	Rarely NULL
<b>UDP</b>		
Source port	2	Random and changing
<b>ICMP</b>		
Data	NA	Usually OS dependant
Sequence number	4	Changing

Last each and every application has interesting field. However, implementation is then dependant of the service to protect and is not likely to be made available for each and every existing application. However, it is still possible to provide a few examples for common applications such as DNS or HTTP.

**Table 14 : Application layer footprint characteristics**

Field	Size (B)	Characterization use
<b>HTTP</b>		
HTTP version	NA	Theoretically 1.0 or 1.1
Method		Usually HEAD, GET or POST
Content-Length		Identifies identical POSTed data
User-Agent		Changes according to the client
Accept		
Accept-Language		
Accept-Charset		
X-Forwarded-For		Identifies clients behind a proxy
<b>DNS</b>		
Transaction ID	2	Random and changing
Question count	2	Depends on the request
DNS Query	NA	Identifies common requests

These fields below are just examples but on top of Layer 3 and 4 information they can add a lot of accuracy in the footprints.

- **Reality of footprinting**

While the theory behind footprinting seems to solve most issues, it remains necessary to evaluate the possibility to accurately fingerprint threats we analyzed in the previous chapters.

Below are some examples of efficient footprints that would accurately characterize the offending traffic;

**Nmap and fping IP sweeps**

Nmap and fping have only one interesting piece of information to feed the footprint : ICMP data. Nmap doesn't generate any data, while fping uses a load of 54 NULL bytes, what is very uncommon.

- Footprint : ICMP DATA SIZE and SOURCE IP and TTL

**THCrut IP sweep**

THCrut always uses the same ICMP sequence number of 0, making it trivial to footprint with an excellent accuracy as it also uses a payload of constant size (8 bytes).

- Footprint : ICMP DATA SIZE and ICMP SEQ and SOURCE IP and TTL

**Nmap TCP scans**

## Mitigating Scanners and Crackers

Nmap main characteristic is that it uses the same sequence number for each packet of a scan. Therefore, in addition to the source IP and the TTL, all detected operations can be accurately stopped.

- Footprint : FLAGS and SEQ and SOURCE IP and TTL

### **Hping TCP scans**

Hping probes have two very discriminative characteristics: a fixed TCP window size, which is not that of the host OS and no TCP options set. Once again an accurate footprint is easy to define based on these characteristics, the source IP and the TTL.

- Footprint : FLAGS and TCP WINDOW and TCP OPTIONS and SOURCE IP and TTL

### **HTTPPrint application identification**

HTTPPrint main characteristic is that it doesn't set any HTTP Header but the URI and sometimes the host. Footprint is therefore trivial thanks to the NULL value of all other HTTP headers.

- Footprint : USER-AGENT and ACCEPT and ACCEPT-LANGUAGE and ACCEPT-CHARSET

### **Hmap application identification**

Hmap has only one characteristic which is its User-Agent, common but quite obsolete. Moreover Accept, Accept-Language and Accept-Charset headers are not set, making the footprint accurate.

- Footprint : USER-AGENT and ACCEPT and ACCEPT-LANGUAGE and ACCEPT-CHARSET

### **Nikto vulnerabilities scan**

Nikto has an interesting User-Agent that uniquely identify the tool : Mozilla/4.75 (Nikto/1.36). On top of it none of the Accept, Accept-Language and Accept-Charset headers are set.

- Footprint : USER-AGENT and ACCEPT and ACCEPT-LANGUAGE and ACCEPT-CHARSET

### **Nessus vulnerability scan**

Nessus looks more subtle as most headers are set. However the combination of an old User-Agent (IE 6.0 under windows), a limited Accept-Language (en) and a unique Accept-Charset (ISO-8859-1,\*,UTF-8) makes fingerprinting very efficient again.

- Footprint : USER-AGENT and ACCEPT and ACCEPT-LANGUAGE and ACCEPT-CHARSET

### **SQLiX attacks**

SQLiX uses a obsolete User-Agent (IE 5.5), and Accept, Accept-Language and Accept-Charset headers are not set. What is more, it sets the HTTP 1.1 TE header [rfc2616], that could be used to be even more accurate in the footprint.

- Footprint : USER-AGENT and ACCEPT and ACCEPT-LANGUAGE and ACCEPT-CHARSET [and TE]

## Mitigating Scanners and Crackers

- **Footprints efficiency**

Based on previous results footprints look efficient in most cases studied in this document. Moreover, examples clearly showed that its efficiency doesn't rely on specific tools and that the technique can be used to prevent many types of threat at many different levels.

However, the main limitation is the necessity to gather an important amount of data that will make statistics reliable enough to create the footprint. Therefore the footprint technique can only be relevant for high volume based threats and will never be applicable to small volume based attacks, such as banner grabbing operations (like nikto application identification) or low noise fingerprinting (like the sinfp technique).

On the other hand, coupled with a detection technique that also relies on high volumes, and this is the case of the error-based statistics, the footprint method is probably one of the most appropriate technique to prevent malicious activities without the needs and the drawbacks of a signature database.

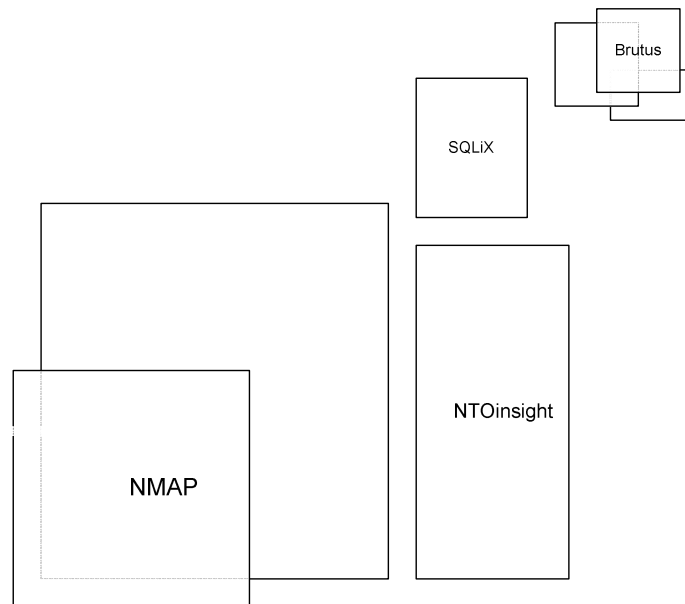
## Conclusion

### Scope of mitigation

Scanners and crackers usually rely on high number of tests. However traffic generated by these probes cannot be efficiently identified by usual network metrics such as bandwidth or packets rate.

Focusing on error rate appeared to be more efficient, as long as errors are effectively generated and based on standard mechanisms. In this case we demonstrated that detection accuracy was excellent for network scans, application identification and vulnerability scans. Based on the fact that most OS fingerprinting tools rely on network scans results, it can be considered, as a side effect, that this category of operation is also efficiently detected by the error-based mechanism, with one noticeable exception : sinfp.

Once high number of data can be gathered and analyzed we observed that footprinting is a valuable solution that will make it possible to block even more threats, exploitation tools being now covered by this technology.



## Mitigating Scanners and Crackers

Back to the threat map, the combination of those two techniques covers most of the issues with a major exception for layer 2 scanning and attacking techniques.

### **Future work and improvements**

Two major fields are to be investigated in order to make this technology even more efficient: sensitivity tuning and application coverage.

The sensitivity can be improved both for the detection engine, in order to detect scans or cracking attempts faster and for the footprinting mechanism. Indeed the need for a large number of packets delays detection and blocking of the offending traffic, making it possible either to get some information or even to compromise a few systems.

Large application footprints coverage is a necessity to cover more accurately scanners and crackers that will usually focus on multiple services. Moreover the capability to efficiently footprint application such as e-mail would lead to another field of application: detection and blocking of mass-mailing misbehavior, such as spam, phishing or even virus propagation, without any signature.

## Appendices

### Appendix A : References

#### Documents

[appdefs] Application definition files for amap – <http://freeworld.thc.org/thc-amap/appdefs.resp>, <http://freeworld.thc.org/thc-amap/appdefs.trig>

[artofscanning] The Art of Scanning – Fyodor - Phrack Magazine #51 – <http://www.phrack.org>

[xprobe-wp] Xprobe v2.0 A “Fuzzy” Approach to Remote Active Operating System - Ofir Arkin, Fyodor Yarochkin - <http://www.sys-security.com/archive/papers/Xprobe2.pdf>

#### Standards

[rfc793] Transmission Control Protocol – RFC 793

[rfc950] Internet Standard Subnetting Procedure – RFC 950

[rfc1323] TCP Extensions for High Performance – RFC 1323

[rfc2616] Hypertext Transfer Protocol -- HTTP/1.1 – RFC 2616

[rfc3168] The Addition of Explicit Congestion Notification (ECN) to IP – RFC 3168

#### Tools

[arpwatch] ARPwatch - <http://ee.lbl.gov/>

[brutus] Brutus AET2 – <http://www.hoobie.net/brutus/>

[ettercap] Ettercap – <http://ettercap.sourceforge.net>

[fping] fping – <http://fping.sourceforge.net>

[httprint] httprint – <http://net-square.com/httprint/>

[hydra] TCH Hydra - <http://freeworld.thc.org/thc-hydra/>

[metasploit] The Metasploit framework – <http://www.metasploit.org>

[nessus] Nessus – <http://www.nessus.org/>

[nmap] NMAP, the network mapper – <http://www.insecure.org/nmap>

[nikto] nikto – <http://www.cirt.net/code/nikto.shtml>

[ntotools] NTOinsight – <http://www.ntobjectives.com>

[sara] SARA, Security Auditor Research Assistant – <http://www-arc.com/sara/>

[siftwms] SIFT Web Method Search – <http://www.sift.com.au>

[sinfp] sinfp - <http://www.gomor.org/cgi-bin/sinfp.pl>

[sipscan] Slp Scanner – <http://www.hackingvoip.com>

[sqlix] SQLiX – [http://www.owasp.org/index.php/Category:OWASP\\_SQLiX\\_Project](http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project)

[sqlninja] SQLNinja

[svmap] svmap - sipvicious tools – <http://sipvicious.org>

## Mitigating Scanners and Crackers

[thcrut] THCrut - aRe yoU There – <http://freeworld.thc.org/thc-rut/>

[venom] – Venom – <http://www.cqure.net>

[voiphopper] – VoIP Hopper - <http://voiphopper.sourceforge.net/>

[wikto] – Wikto - web server assessment tool – <http://www.sensepost.com/research/wikto/>

[xprobe] Xprobe2 – <http://xprobe.sourceforge.net>

[yersinia] – Yersinia - <http://www.yersinia.net/>

## Appendix B : Command Lines

### Layer 2 scans

#### ➤ **THCrut**

ARP scan : #thcrut arp 192.168.205.1-192.168.205.254

### Layer 3 scans

#### ➤ **Fping**

Ping scan : #fping -aA -g 10.0.0.1 10.0.0.255 2>/dev/null

#### ➤ **THCrut**

Ping scan : #thcrut icmp -P 10.0.0.1-10.0.0.255

Netmask scan : #thcrut icmp -A 10.0.0.1-10.0.0.255

#### ➤ **Nmap**

Ping scan : #nmap -sP -PI 10.0.0.0/24

Netmask scan : #nmap -sP -PM 10.0.0.0/24

Timestamp scan : #nmap -sP -PP 10.0.0.0/24

### Layer 4 scans

#### ➤ **Nmap :**

TCP Connect scan : #nmap -P0 -sT -p 1-1024 10.0.0.101

TCP Half scan : #nmap -P0 -sS -p 1-1024 10.0.0.101

TCP FIN scan : #nmap -P0 -sF -p 1-1024 10.0.0.101

TCP XMAS scan : #nmap -P0 -sX -p 1-1024 10.0.0.101

#### ➤ **Hping :**

TCP Half scan : #hping --scan 1-1024 -i u10 -S 10.0.0.101

TCP FIN scan : #hping --scan 1-1024 -i u10 -F 10.0.0.101

TCP XMAS scan : #hping --scan 1-1024 -i u10 -FUP 10.0.0.101

## Mitigating Scanners and Crackers

### OS scans

➤ **Nmap :**

Fingerprinting : #nmap -O -P0 -p 3,443 10.0.0.105

➤ **Xprobe2:**

Fingerprinting : #xprobe2 -p tcp:3:CLOSED -p tcp:443:open -p udp:1:closed -p udp:135:open -p udp:161:open -p tcp:139:open 10.0.0.105 -v -D 1 -D 2 -D 3 -D 4 -D 5

➤ **THCrut**

Fingerprinting : #thcrut discover -O 10.0.0.105

➤ **sinfo :**

Fingerprinting : #sinfo.pl -i 10.0.0.105 -p 80

### Application scans

➤ **Nmap**

Banner grabbing : #nmap -A -P0 -p21,25,80,135,443,3306 10.0.0.105

➤ **Amap**

Fingerprinting : #amap 10.0.0.105 21 25 80 135 443 3306

➤ **httprint**

Fingerprinting : #httprint -h 10.0.0.105 -s signatures.txt

➤ **hmap**

Fingerprinting : #python hmap.py http://10.0.0.105:80

➤ **nikto**

Banner grabbing : #nikto.pl -findonly -host 10.0.0.101

### Vulnerability scans

➤ **ntoinsight**

Vulnerability scanning : #ntoinsight.exe -ntoweb -h 10.0.0.207 -np

➤ **nikto**

Vulnerability scanning : #perl nikto.pl -Cgidirs all -generic -host 10.0.0.207