9

PASSWORD ATTACKS

Passwords are often the path of least resistance on pentesting engagements. A client with a strong security program can fix missing Windows patches and out-of-date software, but the users themselves can't be patched. We'll look at attacking users when we discuss social engineering in Chapter 11, but if we can correctly guess or calculate a user's password, we may be able to avoid involving the user in the attack at all. In this chapter we'll look at how to use tools to automate running services on our targets and sending usernames and passwords. Additionally, we'll study cracking the password hashes we gained access to in Chapter 8.

Password Management

Companies are waking up to the inherent risks of password-based authentication; brute-force attacks and educated guesses are both serious risks to weak passwords. Many organizations use biometric (fingerprint or retinal

scan-based) or two-factor authentication to mitigate these risks. Even web services such as Gmail and Dropbox offer two-factor authentication in which the user provides a password as well as a second value, such as the digits on an electronic token. If two-factor authentication is not available, using strong passwords is imperative for account security because all that stands between the attacker and sensitive data may come down to a simple string. Strong passwords are long, use characters from multiple complexity classes, and are not based on a dictionary word.

The passwords we use in this book are deliberately terrible, but unfortunately, many users don't behave much better when it comes to passwords. Organizations can force users to create strong passwords, but as passwords become more complex, they become harder to remember. Users are likely to leave a password that they can't remember in a file on their computer, in their smartphone, or even on a Post-it note, because it's just easier to keep of track them that way. Of course, passwords that can be discovered lying around in plaintext undermine the security of using a strong password.

Another cardinal sin of good password management is using the same password on many sites. In a worst-case scenario, the CEO's weak password for a compromised web forum might just be the very same one for his or her corporate access to financial documents. Password reuse is something to bear in mind while performing password attacks; you may find the same passwords work on multiple systems and sites.

Password management presents a difficult problem for IT staff and will likely continue to be a fruitful avenue for attackers unless or until password-based authentication is phased out entirely in favor of another model.

Online Password Attacks

Just as we used automated scans to find vulnerabilities, we can use scripts to automatically attempt to log in to services and find valid credentials. We'll use tools designed for automating online password attacks or guessing passwords until the server responds with a successful login. These tools use a technique called *brute forcing*. Tools that use brute forcing try every possible username and password combination, and given enough time, they *will* find valid credentials.

The trouble with brute forcing is that as stronger passwords are used, the time it takes to brute-force them moves from hours to years and even beyond your natural lifetime. We can probably find working credentials more easily by feeding educated guesses about the correct passwords into an automated login tool. Dictionary words are easy to remember, so despite the security warnings, many users incorporate them into passwords. Slightly more security-conscious users might put some numbers at the end of their password or maybe even an exclamation point.

Wordlists

Before you can use a tool to guess passwords, you need a list of credentials to try. If you don't know the name of the user account you want to crack, or you just want to crack as many accounts as possible, you can provide a username list for the password-guessing tool to iterate through.

User Lists

When creating a user list, first try to determine the client's username scheme. For instance, if we're trying to break into employee email accounts, figure out the pattern the email addresses follow. Are they *firstname.lastname*, just a first name, or something else?

You can look for good username candidates on lists of common first or last names. Of course, the guesses will be even more likely to succeed if you can find the names of your target's actual employees. If a company uses a first initial followed by a last name for the username scheme, and they have an employee named John Smith, *jsmith* is likely a valid username. Listing 9-1 shows a very short sample user list. You'd probably want a larger list of users in an actual engagement.

```
root@kali:~# cat userlist.txt
georgia
john
mom
james
```

Listing 9-1: Sample user list

Once you've created your list, save the sample usernames in a text file in Kali Linux, as shown in Listing 9-1. You'll use this list to perform online password attacks in "Guessing Usernames and Passwords with Hydra" on page 202.

Password Lists

In addition to a list of possible users, we'll also need a password list, as shown in Listing 9-2.

```
root@kali:~# cat passwordfile.txt
password
Password
password1
Password1
Password123
password123
```

Listing 9-2: Sample password list

Like our username list, this password list is just a very short example (and one that, hopefully, wouldn't find the correct passwords for too many accounts in the real world). On a real engagement, you should use a much longer wordlist.

There are many good password lists available on the Internet. Good places to look for wordlists include *http://packetstormsecurity.com/Crackers/wordlists/* and *http://www.openwall.com/wordlists/*. A few password lists are also built into Kali Linux. For example, the */usr/share/wordlists* directory contains a file called *rockyou.txt.gz*. This is a compressed wordlist. If you unzip the file with the gunzip Linux utility, you'll have about 140 MB of possible passwords, which should give you a pretty good start. Also, some of the password-cracking tools in Kali come with sample wordlists. For example, the John the Ripper tool (which we'll use in "Offline Password Attacks" on page 203) includes a wordlist at */usr/share/john/password.lst*.

For better results, customize your wordlists for a particular target by including additional words. You can make educated guesses based on information you gather about employees online. Information about spouses, children, pets, and hobbies may put you on the right track. For example, if your target's CEO is a huge Taylor Swift fan on social media, consider adding keywords related to her albums, her music, or her boyfriends. If your target's password is *TaylorSwift13!*, you should be able to confirm it using password guessing long before you have to run a whole precompiled wordlist or a brute-force attempt. Another thing to keep in mind is the language(s) used by your target. Many of your pentesting targets may be global.

In addition to making educated guesses based on information you gather while performing reconnaissance, a tool like the ceWL custom wordlist generator will search a company website for words to add to your wordlist. Listing 9-3 shows how you might use ceWL to create a wordlist based on the contents of *www.bulbsecurity.com*.

```
root@kali:~# cewl --help
cewl 5.0 Robin Wood (robin@digininja.org) (www.digininja.org)

Usage: cewl [OPTION] ... URL
--snip--
--depth x, -d x: depth to spider to, default 2 ❶
--min_word_length, -m: minimum word length, default 3 ❷
--offsite, -o: let the spider visit other sites
--write, -w file: write the output to the file ❸
--ua, -u user-agent: useragent to send
--snip--
URL: The site to spider.
root@kali:~# cewl -w bulbwords.txt -d 1 -m 5 www.bulbsecurity.com ❹
```

Listing 9-3: Using ceWL to build custom wordlists

The command `ceWL --help` lists ceWL's usage instructions. Use the `-d` (depth) option ❶ to specify how many links ceWL should follow on the target website. If you think that your target has a minimum password-size requirement, you might specify a minimum word length to match with the `-m` option ❷. Once you've made your choices, output ceWL's results to a file with the `-w` option ❸. For example, to search *www.bulbsecurity.com* to depth `1` with minimum word length of `5` characters and output the words found to the file *bulbwords.txt*, you would use the command shown at ❹. The resulting file would include all words found on the site that meet your specifications.

Another method for creating wordlists is producing a list of every possible combination of a given set of characters, or a list of every combination of characters for a specified number of characters. The tool Crunch in Kali will generate these character sets for you. Of course, the more possibilities, the more disk space is required for storage. A very simple example of using Crunch is shown in Listing 9-4.

```
root@kali:~# crunch 7 7 AB
Crunch will now generate the following amount of data: 1024 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 128
AAAAAAA
AAAAAAB
--snip--
```

Listing 9-4: Brute-forcing a keyspace with Crunch

This example generates a list of all the possible seven-character combinations of just the characters *A* and *B*. A more useful, but much, much larger example would be entering `crunch 7 8`, which would generate a list of all the possible combinations of characters for a string between seven and eight characters in length, using the default Crunch character set of lowercase letters. This technique is known as *keyspace brute-forcing*. While it is not feasible to try every possible combination of characters for a password in the span of your natural life, it is possible to try specific subsets; for instance, if you knew the client's password policy requires passwords to be at least seven characters long, trying all seven- and eight-character passwords would probably result in cracking success—even among the rare users who did not base their passwords on a dictionary word.

NOTE *Developing a solid wordlist or set of wordlists is a constantly evolving process. For the exercises in this chapter, you can use the short sample wordlist we created in Listing 9-2, but as you gain experience in the field, you'll develop more complex lists that work well on client engagements.*

Now let's see how to use our wordlist to guess passwords for services running on our targets.

Guessing Usernames and Passwords with Hydra

If you have a set of credentials that you'd like to try against a running service that requires a login, you can input them manually one by one or use a tool to automate the process. Hydra is an online password-guessing tool that can be used to test usernames and passwords for running services. (Following the tradition of naming security tools after the victims of Heracles's labors, Hydra is named for the mythical Greek serpent with many heads.) Listing 9-5 shows an example of using Hydra for online password guessing.

```
root@kali:~# hydra -L userlist.txt -P passwordfile.txt 192.168.20.10 pop3
Hydra v7.6 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2015-01-12 15:29:26
[DATA] 16 tasks, 1 server, 24 login tries (l:4/p:6), ~1 try per task
[DATA] attacking service pop3 on port 110
[110][pop3] host: 192.168.20.10   login: georgia   password: password❶
[STATUS] attack finished for 192.168.20.10 (waiting for children to finish)
1 of 1 target successfuly completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2015-01-12 15:29:48
```

Listing 9-5: Using Hydra to guess POP3 usernames and passwords

Listing 9-5 shows how to use Hydra to guess usernames and passwords by running through our username and password files to search for valid POP3 credentials on our Windows XP target. This command uses the `-L` flag to specify the username file, the `-P` for the password list file, and specifies the protocol `pop3`. Hydra finds that user *georgia*'s password is `password` at ❶. (Shame on *georgia* for using such an insecure password!)

Sometimes you'll know that a specific username exists on a server, and you just need a valid password to go with it. For example, we used the `SMTP VRFY` verb to find valid usernames on the SLMail server on the Windows XP target in Chapter 6. As you can see in Listing 9-6, we can use the `-l` flag instead of `-L` to specify one particular username. Knowing that, let's look for a valid password for user *georgia* on the `pop3` server.

```
root@kali:~# hydra -l georgia -P passwordfile.txt 192.168.20.10 pop3
Hydra v7.6 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only
[DATA] 16 tasks, 1 server, 24 login tries (l:4/p:6), ~1 try per task
[DATA] attacking service pop3 on port 110
[110][pop3] host: 192.168.20.10   login: georgia   password: password❶
[STATUS] attack finished for 192.168.20.10 (waiting for children to finish)
1 of 1 target successfuly completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2015-01-07 20:22:23
```

Listing 9-6: Using a specific username with Hydra

Hydra found *georgia*'s password to be *password* ❶.

Now, in Listing 9-7, we'll use our credentials to read *georgia*'s email.

```
root@kali:~# nc 192.168.20.10 pop3
+OK POP3 server xpvictim.com ready <00037.23305859@xpvictim.com>
```

```
USER georgia
+OK georgia welcome here
PASS password
+OK mailbox for georgia has 0 messages (0 octets)
```

Listing 9-7: Using Netcat to log in with guessed credentials

Specify the `pop3` protocol, and provide the username and password when prompted. (Unfortunately, there are no love letters in this particular inbox.) Hydra can perform online password guessing against a range of services. (See its manual page for a complete list.) For example, here we use the credentials we found with Hydra to log in with Netcat.

Keep in mind that most services can be configured to lock out accounts after a certain number of failed login attempts. There are few better ways to get noticed by a client's IT staff than suddenly locking out several user accounts. Logins in rapid succession can also tip off firewalls and intrusion-prevention systems, which will get your IP address blocked at the perimeter. Slowing down and randomizing scans can help with this, but there is, of course, a tradeoff: Slower scans will take longer to produce results.

One way to avoid having your login attempts noticed is to try to guess a password before trying to log in, as you'll learn in the next section.

Offline Password Attacks

Another way to crack passwords (without being discovered) is to get a copy of the password hashes and attempt to reverse them back to plaintext passwords. This is easier said than done because hashes are designed to be the product of a one-way hash function: Given an input, you can calculate the output using the hash function, but given the output, there is no way to reliably determine the input. Thus, if a hash is compromised, there should be no way to calculate the plaintext password. We can, however, guess a password, hash it with the one-way hash function, and compare the results to the known hash. If the two hashes are the same, we've found the correct password.

NOTE *As you'll learn in "LM vs. NTLM Hashing Algorithms" on page 208, not all password hashing systems have stood the test of time. Some have been cracked and are no longer considered secure. In these cases, regardless of the strength of the password chosen, an attacker with access to the hashes will be able to recover the plaintext password in a reasonable amount of time.*

Of course, it's even better if you can get access to passwords in plaintext and save yourself the trouble of trying to reverse the cryptography, but often the passwords you encounter will be hashed in some way. In this section we'll focus on finding and reversing password hashes. If you stumble upon a program configuration file, database, or other file that stores passwords in plaintext, all the better.

But before we can try to crack password hashes, we have to find them. We all hope that the services that store our passwords do a good job of

protecting them, but that's never a given. It only takes one exploitable flaw or a user who falls victim to a social-engineering attack (discussed in Chapter 11) to bring down the whole house of cards. You'll find plenty of password hashes lying around sites like Pastebin, remnants from past security breaches.

In Chapter 8, we gained access to some password hashes on the Linux and Windows XP targets. Having gained a Meterpreter session with system privileges on the Windows XP system via the *windows/smb/ms08_067_netapi* Metasploit module, we can use the `hashdump` Meterpreter command to print the hashed Windows passwords, as shown in Listing 9-8.

```
meterpreter > hashdump
Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c:::
georgia:1003:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:df40c521ef762bb7b9767e30ff112a3c:938ce7d211ea733373bcfc3e6fbb3641:::
secret:1004:e52cac67419a9a22664345140a852f61:58a478135a93ac3bf058a5ea0e8fdb71:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:bc48640a0fcb55c6ba1c9955080a52a8:::
```

Listing 9-8: Dumping password hashes in Meterpreter

Save the output of the hashdump to a file called *xphashes.txt*, which we will use in "John the Ripper" on page 210.

In Chapter 8 we also downloaded backups of the SAM and SYSTEM hives using the local file inclusion issue in Zervit 0.4 on the Windows XP system. We used this same issue to download the configuration file for the FileZilla FTP server, which contained passwords hashed with the MD5 algorithm. On the Linux target, the Vsftpd smiley-face backdoor gave us root privileges, and thus we can access to the file */etc/shadow*, which stores Linux password hashes. We saved the password for user *georgia* to the file *linuxpasswords.txt*.

Recovering Password Hashes from a Windows SAM File

The SAM file stores hashed Windows passwords. Though we were able to use Meterpreter to dump the password hashes from the Windows XP system (as shown previously), sometimes you'll be able to get only the SAM file.

We weren't able to get access to the primary SAM file through the Zervit 0.4 vulnerability, but we were able to download a backup copy from the *C:\Windows\repair* directory using a local file-inclusion vulnerability. But when we try to read the SAM file (as shown in Listing 9-9), we don't see any password hashes.

```
root@bt:~# cat sam
regf     P P5gfhbinÐÐÐÐnk,ÐuÐÐÐÐÐ ÐÐÐÐ ÐÐÐÐÐÐÐÐÐxÐÐÐÐSAMXÐÐÐskx x  Ð ÐpÐµ\µ?
?   µ   µ
                                    ÐÐÐÐnk LÐÐÐÐ  ÐBÐÐÐÐ Ðx ÐÐÐÐÐSAMÐÐÐÐskxx7d
ÐHXµ4µ?          ÐÐÐÐvk Ð CPÐÐÐ Ð  µÐxÐµDOÐµ    ÐµÐÐ  4µ1   ?         ÐÐÐÐÐ
ÐÐÐÐlf   SAMÐÐÐÐnk ÐuÐÐÐÐÐ   H#ÐÐÐÐ Px ÐÐÐÐDomainsÐÐÐÐvkÐÐÐÐÐ8lf ÐDomaÐÐÐÐnk
\ÐÐJÐÐÐ ÐÐÐÐÐÐOx ÐÐÐÐ( AccountÐÐÐÐvk ÐÐ
--snip--
```

Listing 9-9: Viewing the SAM file

The SAM file is obfuscated because the Windows Syskey utility encrypts the password hashes inside the SAM file with 128-bit Rivest Cipher 4 (RC4) to provide additional security. Even if an attacker or pentester can gain access to the SAM file, there's a bit more work to do before we can recover the password hashes. Specifically, we need a key to reverse the encrypted hashes.

The encryption key for the Syskey utility is called the *bootkey*, and it's stored in the Windows SYSTEM file. You'll find a copy of the SYSTEM file in the *C:\Windows\repair* directory where we found the backup SAM file. We can use a tool in Kali called Bkhive to extract the Syskey utility's bootkey from the SYSTEM file so we can decrypt the hashes, as shown in Listing 9-10.

```
root@kali:~# bkhive system xpkey.txt
bkhive 1.1.1 by Objectif Securite
http://www.objectif-securite.ch
original author: ncuomo@studenti.unina.it

Root Key : $$$PROTO.HIV
Default ControlSet: 001
Bootkey: 015777ab072930b22020b999557f42d5
```

Listing 9-10: Using Bkhive to extract the bootkey

Here we use Bkhive to extract the bootkey by passing in the SYSTEM file *system* (the file we downloaded from the repair directory using the Zervit 0.4 directory traversal) as the first argument and extracting the file to *xpkey.txt*. Once we have the bootkey, we can use Samdump2 to retrieve the password hashes from the SAM file, as shown in Listing 9-11. Pass Samdump2 the location of the SAM file and the bootkey from Bkhive as arguments, and it will use the bootkey to decrypt the hashes.

```
root@kali:~# samdump2 sam xpkey.txt
samdump2 1.1.1 by Objectif Securite
http://www.objectif-securite.ch
original author: ncuomo@studenti.unina.it

Root Key : SAM
Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:df40c521ef762bb7b9767e30ff112a3c:938ce7d211ea733373bcfc3e6fbb3641:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:bc48640a0fcb55c6ba1c9955080a52a8:::
```

Listing 9-11: Using Samdump2 to recover Windows hashes

Now compare these hashes to those found with the `hashdump` command in an active Meterpreter session from Listing 9-8. (A Meterpreter session with sufficient privileges can dump password hashes on the fly without requiring us to download the SAM and SYSTEM files.) Notice that our hash list in Listing 9-11 lacks entries for the users *georgia* or *secret*. What happened?

When using the Zervit directory traversal, we weren't able to access the main SAM file at *C:\Windows\System32\config* and instead downloaded a backup from *C:\Windows\repair\sam*. These users must have been created

after the SAM file backup was created. We do have a password hash for the *Administrator* user, though. Though not complete or fully up-to-date, we may still be able to use cracked hashes from this backup SAM to log in to the systems.

Now let's look at another way to access password hashes.

Dumping Password Hashes with Physical Access

On some engagements, you'll actually have physical access to user machines, with so-called physical attacks in scope. While having physical access may not appear very useful at first, you may be able to access the password hashes by restarting a system using a Linux Live CD to bypass security controls. (We'll use a Kali ISO image, though other Linux Live CDs such as Helix or Ubuntu will work. We used a prebuilt Kali virtual machine in Chapter 1. To get a standalone ISO of Kali, go to *http://www.kali.org*.) When you boot a machine with a Live CD, you can mount the internal hard disk and gain access to all files, including the SAM and SYSTEM files. (When Windows boots, there are certain security controls in place to stop users from accessing the SAM file and dumping password hashes, but these aren't active when the filesystem is loaded in Linux.)

Our Windows 7 virtual machine, with its solid external security posture, has been a bit neglected in these last few chapters. Let's dump its hashes using a physical attack. First, we'll point our virtual machine's optical drive to a Kali ISO file, as shown in Figure 9-1 (for VMware Fusion). In VMware Player, highlight your Windows 7 virtual machine, right-click it and choose **Settings**, then choose **CD/DVD (SATA)** and point to the ISO in the Use ISO Image field on the right side of the page.

Figure 9-1: Setting our Windows 7 virtual machine to boot from the Kali ISO file

By default, VMware will boot up the virtual machine so quickly that it will be difficult to change the BIOS settings to boot from the CD/DVD drive instead of the hard disk. To fix this, we'll add a line to the VMware configuration file (*.vmx*) to delay the boot process at the BIOS screen for a few seconds.

1. On your host machine, browse to where you saved your virtual machines. Then, in the folder for the Windows 7 target, find the *.vmx* configuration file, and open it in a text editor. The configuration file should look similar to Listing 9-12.

```
.encoding = "UTF-8"
config.version = "8"
virtualHW.version = "9"
vcpu.hotadd = "TRUE"
scsi0.present = "TRUE"
scsi0.virtualDev = "lsilogic"
--snip--
```

Listing 9-12: VMware configuration file (.vmx)

2. Add the line **`bios.bootdelay = 3000`** anywhere in the file. This tells the virtual machine to delay booting for 3000 ms, or three seconds, enough time for us to change the boot options.
3. Save the *.vmx* file, and restart the Windows 7 target. Once you can access the BIOS, choose to boot from the CD drive. The virtual machine should start the Kali ISO. Even though we're booted into Kali, we can mount the Windows hard disk and access files, bypassing the security features of the Windows operating system.

Listing 9-13 shows how to mount the file system and dump the password hashes.

```
root@kali:# ❶mkdir -p /mnt/sda1
root@kali:# ❷mount /dev/sda1 /mnt/sda1
root@kali:# ❸cd /mnt/sda1/Windows/System32/config/
root@kali:/mnt/sda1/Windows/System32/config  bkhive SYSTEM out
root@kali:/mnt/sda1/Windows/System32/config  samdump2 SAM out
samdump2 1.1.1 by Objectif Securite
http://www.objectif-securite.ch
original author: ncuomo@studenti.unina.it

Root Key : CMI-CreateHive{899121E8-11D8-41B6-ACEB-301713D5ED8C}
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Georgia Weidman:1000:aad3b435b51404eeaad3b435b51404ee:8846f7eaee8fb117ad06bdd830b75B6c:::
```

Listing 9-13: Dumping Windows hashes with a Linux Live CD

We create a directory where we can mount our Windows filesystem with the `mkdir` command at ❶. Next, we use `mount` ❷ to mount the Windows filesystem (*/dev/sda1*) in the newly created directory (*/mnt/sda1*), which means that the target's C drive is effectively at */mnt/sda1*. The SAM and SYSTEM files in Windows are in the *C:\Windows\System32\config* directory, so we change directories to */mnt/sda1/Windows/System32/config* to access these files using

cd ❸, at which point we can use Samdump2 and Bkhive against the SAM and SYSTEM files without first saving these files and moving them to our Kali system.

Once again we've managed to get access to password hashes. We now have hashes for our Windows XP target, our Windows 7 target, our Linux target, and the FileZilla FTP server on the Windows XP target.

NOTE *In Chapter 13, we'll explore some tricks for using password hashes to authenticate without the need for access to the plaintext passwords, but usually, in order to use these hashes, we'll need to reverse the cryptographic hash algorithms and get the plaintext passwords. The difficulty of this depends on the password-hashing algorithm used as well as the strength of the password used.*

LM vs. NTLM Hashing Algorithms

Listing 9-14 compares the two password hash entries. The first one belongs to the *Administrator* account on Windows XP, which we found with `hashdump` in Meterpreter, and the second is Georgia Weidman's account from Windows 7, which we found with physical access in the previous section.

```
Administrator❶:500❷:e52cac67419a9a224a3b108f3fa6cb6d❸:8846f7eaee8fb117ad06bdd830b7586c❹
Georgia Weidman❶:1000❷:aad3b435b51404eeaad3b435b51404ee❸:8846f7eaee8fb117ad06bdd830b7586c❹
```

Listing 9-14: Dumping Windows hashes with a Linux Live CD

The first field in the hashes is the username ❶; the second is the user ID ❷; the third is the password hash in LAN Manager (LM) format ❸; and the fourth is the NT LAN Manager (NTLM) hash ❹. LM Hash was the primary way to hash passwords on Microsoft Windows up to Windows NT, but it's a cryptographically unsound method that makes it possible to discover the correct plaintext password for an LM hash, regardless of a password's length and complexity. Microsoft introduced NTLM hashing to replace LM hash, but on Windows XP, passwords are stored in both LM and NTLM formats by default. (Windows 7 opts exclusively for the more secure NTLM hash.)

In the hashes in Listing 9-14, because both passwords are the string *password*, the NTLM hash entries for each account are identical, but the LM hash fields are different. The first entry has the value `e52cac67419a9a224a3b108f3fa6cb6d`, whereas the Windows 7 entry has `aad3b435b51404eeaad3b435b51404ee`, which is LM hash-speak for empty. The inclusion of the LM hash entry will make cracking the hashes much simpler. In fact, any LM-hashed password can be brute-forced in minutes to hours. In contrast, our ability to crack the NTLM hashes will depend on both our ability to guess and the length and complexity of the password. If the hashing function is cryptographically sound, it could take years, decades, or more than your lifetime to try every possible password.

The Trouble with LM Password Hashes

When you see LM hashes on a pentest, you can be sure that the plaintext password is recoverable from the password hash. However, one-way hash functions can't be reversed. Complex math is used to develop algorithms that make it impossible to discover the original plaintext password value that was hashed, given the password hash. But we *can* run a plaintext password guess through the cryptographic hashing function and compare the results to the hash we're trying to crack; if they're the same, we've found the correct password.

The following issues contribute to the insecurity of LM hashes:

- Passwords are truncated at 14 characters.
- Passwords are converted to all uppercase.
- Passwords of fewer than 14 characters are null-padded to 14 characters.
- The 14-character password is broken into two seven-character passwords that are hashed separately.

Why are these characteristics so significant? Say we start with a complex, strong password like this:

```
T3LF23!+?sRty$J
```

This password has 15 characters from four classes, including lowercase letters, uppercase letters, numbers, and symbols, and it's not based on a dictionary word. However, in the LM hash algorithm, the password is truncated to 14 characters like this:

```
T3LF23!+?sRty$
```

Then the lowercase letters are changed to uppercase:

```
T3LF23!+?SRTY$
```

Next, the password is split into two seven-character parts. The two parts are then used as keys to encrypt the static string `KGS!@#$%` using the Data Encryption Standard (DES) encryption algorithm:

```
T3LF23!     +?SRTY$
```

The resulting eight-character ciphertexts from the encryption are then concatenated to make the LM hash.

To crack an LM hash, we just need to find seven characters, all uppercase, with perhaps some numbers and symbols. Modern computing hardware can try every possible one- to seven-character combination, encrypt the string `KGS!@#$%`, and compare the resulting hash to a given value in a matter of minutes to hours.

John the Ripper

One of the more popular tools for cracking passwords is John the Ripper. The default mode for John the Ripper is brute forcing. Because the set of possible plaintext passwords in LM hash is so limited, brute forcing is a viable method for cracking any LM hash in a reasonable amount of time, even with our Kali virtual machine, which has limited CPU power and memory.

For example, if we save the Windows XP hashes we gathered earlier in this chapter to a file called *xphashes.txt*, then feed them to John the Ripper like this, we find that John the Ripper can run through the entire set of possible passwords and come up with the correct answer, as shown in Listing 9-15.

```
root@kali: john xphashes.txt
Warning: detected hash type "lm", but the string is also recognized as "nt"
Use the "--format=nt" option to force loading these as that type instead
Loaded 10 password hashes with no different salts (LM DES [128/128 BS SSE2])
                 (SUPPORT_388945a0)
PASSWOR          (secret:1)
                 (Guest)
PASSWOR          (georgia:1)
PASSWOR          (Administrator:1)
D                (georgia:2)
D                (Administrator:2)
D123             (secret:2)
```

Listing 9-15: Cracking LM hashes with John the Ripper

John the Ripper cracks the seven-character password hashes. In Listing 9-15, we see that *PASSWOR* is the first half of the user *secret*'s password. Likewise, it's the first half of the password for *georgia* and *Administrator*. The second half of *secret*'s password is *D123*, and *georgia* and *Administrator*'s are *D*. Thus, the complete plaintext of the LM-hashed passwords are *PASSWORD* for *georgia* and *Administrator* and *PASSWORD123* for *secret*. The LM hash doesn't tell us the correct case for a password, and if you try logging in to the Windows XP machine as *Administrator* or *georgia* with the password *PASSWORD* or the account *secret* with *PASSWORD123*, you will get a login error because LM hash does not take into account the correct case of the letters in the password.

To find out the correct case of the password, we need to look at the fourth field of the NTLM hash. John the Ripper noted in the example in Listing 9-15 that NTLM hashes were also present, and you can use the flag `--format=nt` to force John the Ripper to use those hashes (we don't have LM hashes for Windows 7, so we will have to crack Windows 7 passwords with a wordlist since brute forcing the NTLM hashes would likely take too long).

Cracking Windows NTLM hashes is nowhere near as easy as cracking LM ones. Although a five-character NTLM password that uses only lowercase letters and no other complexity could be brute-forced as quickly as an LM hash, a 30-character NTLM password with lots of complexity could

take many years to crack. Trying every possible character combination of any length, hashing it, and comparing it to a value could go on forever until we happened to stumble upon the correct value (only to find out that the user has since changed his or her password).

Instead of attempting to brute-force passwords, we can use wordlists containing known passwords, common passwords, dictionary words, combinations of dictionary words padded with numbers and symbols at the end, and so on. (We'll see an example of using a wordlist with John the Ripper in "Cracking Linux Passwords" on page 212).

A REAL-WORLD EXAMPLE

Legacy password hashing once made all the difference on one of my pentests. The domain controller was Windows Server 2008, with a strong security posture. The workstations throughout the enterprise were reasonably secure, too, having recently been upgraded to fully patched Windows 7 systems. There was, however, one promising light in the dark: a Windows 2000 box that was missing several security patches. I was able to quickly gain system privileges on the machine using Metasploit.

The trouble was that, while on paper, the penetration test was now a success, compromising the machine had gained me next to nothing. The system contained no sensitive files, and it was the only machine on this particular network, isolated from the new, updated Windows domain. It had all the trappings of a domain controller, except it had no clients. All of the other machines in the environment were members of the new Windows 2008 domain controller's domain. Though technically I was now a domain administrator, I was no further along on the pentest than I was before I found the Windows 2000 machine.

Since this was the domain controller, the domain user password hashes were included locally. Windows 2000, like Windows XP, stored the LM hashes of passwords. The client's old domain administrator password was strong; it had about 14 characters; included uppercase letters, lowercase letters, numbers, and symbols; and was not based on a dictionary word. Fortunately, because it was LM hashed, I was able to get the password back in a matter of minutes.

What do you think the domain administrator's password was on the new domain? You guessed it. It was the same as the domain administrator's password on the old domain. The Windows 2000 box had not been used in over six months, but it was still running, and it used an insecure hashing algorithm. Also, the client wasn't changing their passwords regularly. These two things combined to bring down what was otherwise a strong security posture. I was able to access every system in the environment just by logging in with the domain administrator password I found on the compromised Windows 2000 system.

Cracking Linux Passwords

We can also use John the Ripper against the Linux password hashes we dumped after exploiting the Vsftpd server backdoor in Chapter 8, as shown in Listing 9-16.

```
root@kali# cat linuxpasswords.txt
georgia:$1$CNp3mty6$lRWcTO/PVYpDKwyaWWkSg/:15640:0:99999:7:::
root@kali# johnlinuxpasswords.txt --wordlist=passwordfile.txt
Loaded 1 password hash (FreeBSD MD5 [128/128 SSE2 intrinsics 4x])
password         (georgia)
guesses: 1  time: 0:00:00:00 DONE (Sun Jan 11 05:05:31 2015)  c/s: 100
trying: password - Password123
```

Listing 9-16: Cracking Linux hashes with John the Ripper

User *georgia* has an MD5 hash (we can tell from the `$1$` at the beginning of the password hash). MD5 can't be brute-forced in a reasonable amount of time. Instead, we use a wordlist with the `--wordlist` option in John the Ripper. John the Ripper's success at cracking the password depends on the inclusion of the correct password in our wordlist.

MANGLING WORDLISTS WITH JOHN THE RIPPER

When required by a password policy to include a number and/or a symbol in a password, many users will just tack them on to the end of a dictionary word. Using John the Ripper's rules functionality, we can catch this and other common mutations that may slip by a simple wordlist. Open the John the Ripper configuration file at */etc/john/john.conf* in an editor and search for *List.Rules:Wordlist*. Beneath this heading, you can add mangling rules for the wordlist. For example, the rule `$[0-9]$[0-9]$[0-9]` will add three numbers to the end of each word in the wordlist. You can enable rules in John the Ripper by using the flag `--rules` at the command line. More information on writing your own rules can be found at *http://www.openwall.com/john/doc/RULES.shtml*.

Cracking Configuration File Passwords

Finally, let's try to crack the MD5 hashed passwords we found in the FileZilla FTP server configuration file we downloaded with the Zervit 0.4 file inclusion vulnerability. As you'll see, sometimes we don't even need to crack a password hash. For example, try entering the hash for the user *georgia*, *5f4dcc3b5aa765d61d8327deb882cf99*, into a search engine. The first few hits confirm that *georgia*'s password is *password*. Additionally, searching tells us that the account *newuser* is created when a FileZilla FTP server is installed with the password *wampp*.

Now try logging in to the Windows XP target's FTP server with these credentials. Sure enough, login is successful. The administrator of this system forgot to change the default password for the built-in FTP account. If we were not able to recover the plaintext passwords this easily, we could again use John the Ripper with a wordlist, as discussed previously.

Rainbow Tables

Rather than taking a wordlist, hashing each entry with the relevant algorithm, and comparing the resulting hash to the value to be cracked, we can speed up this process considerably by having our wordlist prehashed. This, of course, will take storage space—more with longer hash lists, and approaching infinity as we try to store every possible password hash value for brute forcing.

A set of precomputed hashes is known as a *rainbow table*. Rainbow tables typically hold every possible hash entry for a given algorithm up to a certain length with a limited character set. For example, you may have a rainbow table for MD5 hashes that contains all entries that are all lowercase letters and numbers with lengths between one and nine. This table is about 80 GB—not so bad with today's price of storage, but keep in mind this is only a very limited amount of the possible keyspace for MD5.

Given its limited keyspace (discussed previously), an LM hash appears to be an ideal candidate for using rainbow tables. A full set of LM hash rainbow tables is about 32 GB.

You can download pregenerated sets of hashes from *http://project-rainbowcrack.com/table.htm*. The tool Rcrack in Kali can be used to sift through the rainbow tables for the correct plaintext.

Online Password-Cracking Services

The current hip thing to do in IT is to move things to the cloud, and password cracking is no different. By leveraging multiple high-spec machines, you can get faster, more comprehensive results than you could with just a virtual machine on your laptop. You can, of course, set up up your own high-powered machines in the cloud, create your own wordlists, and so on, but there are also online services that will take care of this for you for a fee. For example, *https://www.cloudcracker.com/* can crack NTLM Windows hashes, SHA-512 for Linux, WPA2 handshakes for wireless, and more. You simply upload your password hash file, and the cracker does the rest.

Dumping Plaintext Passwords from Memory with Windows Credential Editor

Why bother cracking password hashes if we can get access to plaintext passwords? If we have access to a Windows system, in some cases we can pull plaintext passwords directly from memory. One tool with this functionality is the Windows Credential Editor (WCE). We can upload this tool to an exploited target system, and it will pull plaintext passwords from the Local

Security Authority Subsystem Service (LSASS) process in charge of enforcing the system's security policy. You can download the latest version of WCE from *http://www.ampliasecurity.com/research/wcefaq.html*. An example of running WCE is shown in Listing 9-17.

```
C:\>wce.exe -w
wce.exe -w
WCE v1.42beta (Windows Credentials Editor) - (c) 2010-2013 Amplia Security - by Hernan Ochoa
(hernan@ampliasecurity.com)
Use -h for help.

georgia\BOOKXP:password
```

Listing 9-17: Running WCE

Here WCE found the plaintext of the user *georgia*'s password. The downside to this attack is that it requires a logged-in user for the password to be stored in memory. Even if you were able to get a plaintext password or two with this method, it is still worth dumping and attempting to crack any password hashes you can access.

Summary

Reversing password hashes is an exciting field, and as the speed of hardware increases, it becomes possible to crack stronger hashes faster. Using multiple CPUs and even the graphics processing units (GPUs) on video cards, password crackers can try many hashes very quickly. Our virtual machines don't have much processing power, but even your average modern laptop is much faster than the machines that were used for password cracking just a few short years ago. The cutting edge of password cracking these days is taking to the cloud and harnessing multiple top-spec cloud servers for cracking. You'll even find some cloud-based password-cracking services.

As you've seen in this chapter, using information gathered from successful exploits in Chapter 8, we've managed to reverse password hashes to recover plaintext passwords for some services and the systems themselves. Having managed to get a foothold on the systems, let's look at some advanced attack methods that can help us if we can't find anything vulnerable when listening on the network. We still have the Windows 7 machine to exploit, after all.