v4

Penetration Testing Student

System Attacks

Section 03 | Module 05

Table of Contents

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ What malware is and the common types
- ✓ Methods of attacking passwords
- ✓ What a buffer overflow is and why it is a serious security risk

5.1

Malware

5.1 Malware

How does this support my pentesting career?

- Ability to use the right malware incarnation during an engagement
- Knowledgeable of how to use malware while keeping your test under the rules of engagement
- Ability to maintain access to a compromised machine

5.1 Malware

Malware, short for "**Mal**icious soft**ware**", is any software used to misuse computer systems with the intent to:

- Cause denial of service
- Spy on users activity
- Get unauthorized control over one or more computer systems
- Cause other malicious activities

5.1 Malware

In this chapter, you will see a complete representation of the various types of malware used by cybercriminals.

Moreover, you will learn how to use some malware incarnations during a **penetration test engagement**.

5.1 Malware

Malware classification is based on the behavior of the software, rather than the malicious features it provides.

- Virus
- Trojan Horses
- Rootkit
- Bootkit
- Backdoors
- Adware
- Spyware
- Greyware
- Dialer
- Key-logger
- Botnet
- Ransomware
- Data-Stealing Malware
- Worm

5.1.1 Virus

A **computer virus** is a small piece of code that spreads from computer to computer, without any direct action or authorization by the owners of the infected machines.

Viruses usually copy themselves to special sections of the hard disk, inside legitimate programs or documents. They then run every time an infected program or file is opened.

5.1.2 Trojan Horse

A **Trojan horse,** as the name suggests, is a malware that comes embedded in a seemingly harmless file such as an executable, an MS Office document, a screen saver or a PDF file.

When a user opens the malicious file, they fall victim to the malware.

5.1.2 Trojan Horse

The most common Trojan horses used by penetration testers are **backdoors**; this kind of Trojan horse lets an attacker get a shell on the infected system.

5.1.3 Backdoor

Backdoors are software made by two components: a server and a backdoor client.

The Backdoor server runs on the victim machine listening on the network and accepting connections. The client usually runs on the attacker machine, and it is used to connect to the backdoor to control it.

5.1.3 Backdoor

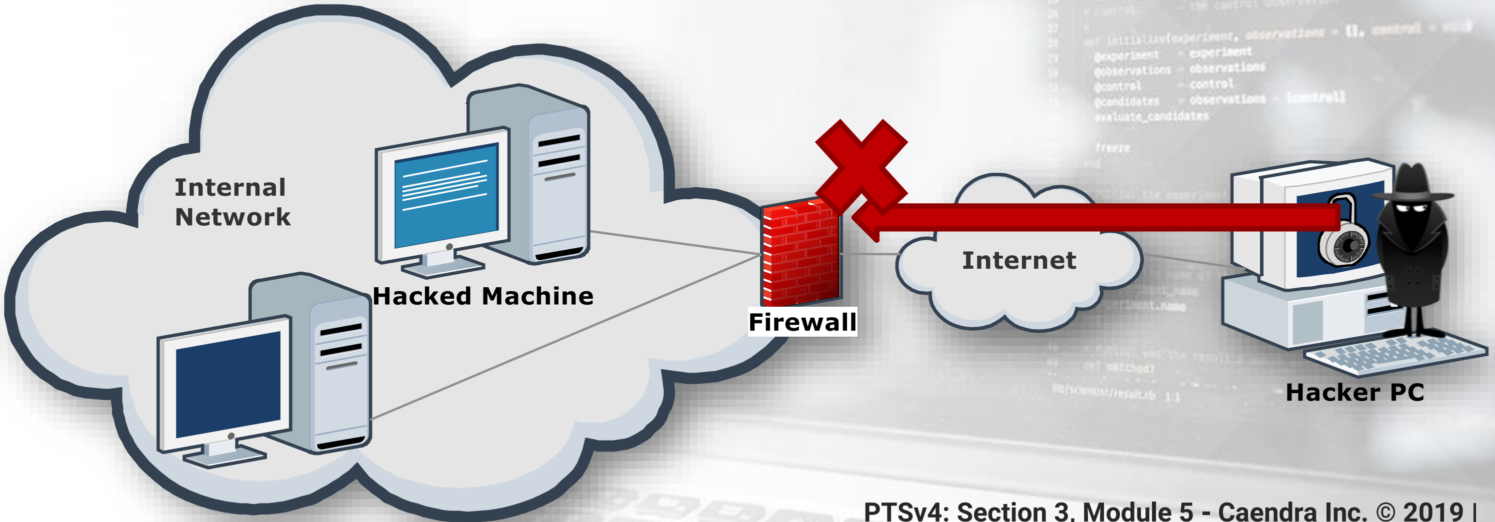
NetBus and **SubSeven** are very famous, old school backdoors; they allow the attacker to browse the victim’s hard drive, upload and download files, execute programs and perform a number of other activities.

After installing and connecting to a backdoor, a penetration tester gets full control over the remote host.

5.1.3.1 Firewalls vs. Backdoors

But what if a wise system administrator configures the network firewall to **block connections** from the internet to internal machines?

The attacker will not be able to connect to the backdoor!!!



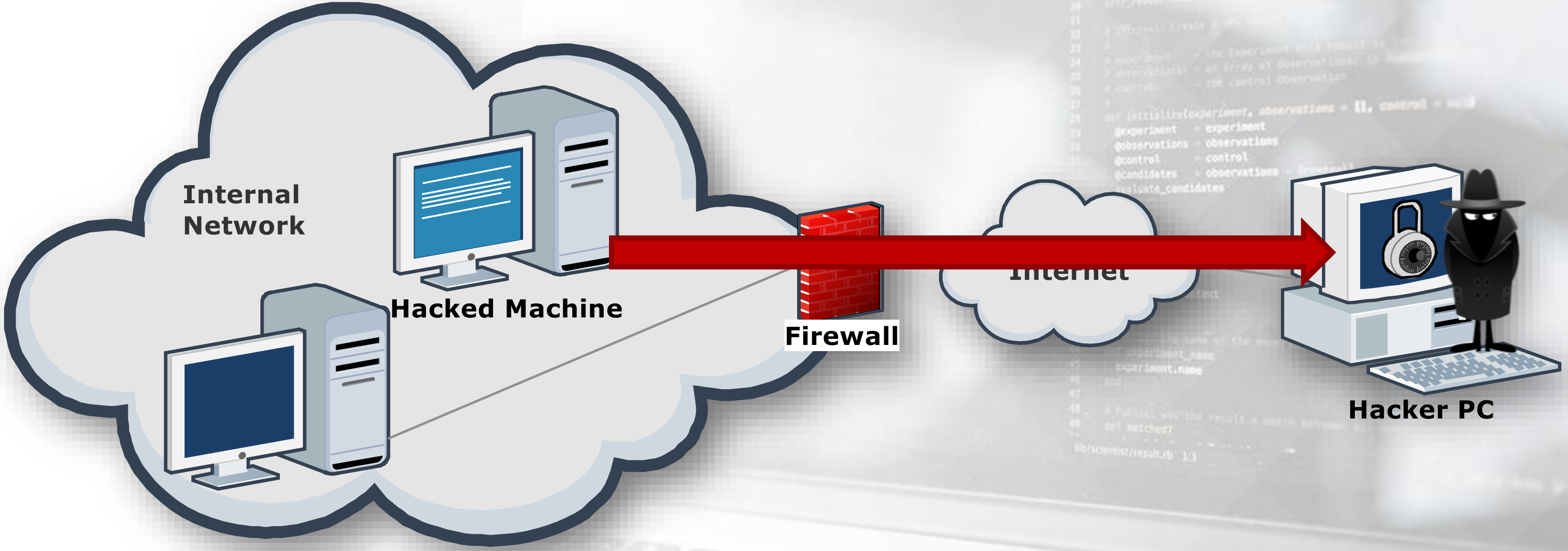
5.1.3.1 Firewalls vs. Backdoors

Moreover, connection tries from the Internet to an internal machine will raise some alarms.

A well-done backdoor would use any means to make the traffic look legitimate; the easiest way to achieve this is by using a **Connect-back Backdoor**.

5.1.3.2 Firewalls vs. Connect-Back Backdoors

A connect-back backdoor, or reverse backdoor, is a common mechanism to bypass firewalls.



5.1.3.2 Firewalls vs. Connect-Back Backdoors

Instead of having the victim machine act as a server and listening to the client's command, it acts as a client and connects back to the penetration tester's machine.

The attacker machine would listen on a port that is known to be commonly allowed on most of the firewalls, such as port 80 (the web server port).

5.1.3.2 Firewalls vs. Connect-Back Backdoors

A firewall cannot tell the difference between a user surfing the web and a backdoor connecting back to the attacker's machine!

5.1.4 Rootkit

A **rootkit** is a malware which is designed to hide itself from users and the antivirus program in order to completely subvert the OS functioning.

It lets an attacker maintain privileged access to the victim machine without being noticed.

5.1.5 Bootkit

Bootkits are rootkits which circumvent OS protection mechanisms by executing during the bootstrap phase.

They start before the operating system, so they get complete control over the machine and the OS.

5.1.6 Adware

Adware is annoying software that shows advertisements to computer users.

5.1.7 Spyware

Spyware is software used to collect information about users' activity. Spyware collects information such as:

- The OS installed on a machine
- Visited websites
- Passwords

The information is sent back to a log collection server controlled by the attacker.

5.1.8 Greyware

Greyware is a general term used to indicate Malware which does not fall under a specific category.

For example, it can be either spyware, adware or both.

5.1.9 Dialer

A **Dialer** is a software that tries to dial numbers on dial-up connections in order to collect money from the victim's phone bill.

Nowadays, dialers target **smartphones**.

5.1.10 Keylogger

A keylogger is a special software which records every keystroke on the remote victim machine.

Operations performed by keyloggers are:

- Recording keystrokes
- Recording the window name where the victim user was typing
- Saving the keystrokes in a log file on the victim machine
- Sending the logs to a server controlled by the penetration tester

5.1.10 Keylogger

Keyloggers are subject to the same restrictions that firewalls pose to backdoors.

If configured wisely, the traffic they generate should not be stopped by a firewall.

5.1.10 Keylogger

When a keylogger runs on a remote machine, login information, emails sent, documents typed and chats get recorded.

Login information can be used to exploit systems, while chat or email information can be used to mount targeted and social engineering attacks.

5.1.10 Keylogger

You have just seen how software keyloggers work. There are also:

- **Hardware keyloggers**
- **Rootkit keyloggers**, which are stealthy and more invisible to the victim user than software keyloggers

5.1.10.1 Hardware Keylogger

Hardware keyloggers are small devices you can install between a keyboard and a computer.

They log keystrokes into an internal memory. An attacker needs two trips to the victim machine to exploit a hardware keylogger: one to install it and one to retrieve it.

5.1.10.1 Hardware Keylogger

Hardware keyloggers are less common than software ones, but they can be used by a penetration tester while performing physical security tests.

Unauthorized access to laboratories or offices may allow a malicious user to use these devices and record proprietary information.

5.1.10.2 Rootkit Keylogger

Rootkit keyloggers are software keyloggers working at the Kernel level by **hijacking the operating system APIs** to record keystrokes.

Every time a key is pressed on a keyboard, a particular function of the OS Kernel is called through a mechanism called an interrupt.

5.1.10.2 Rootkit Keylogger

There are many different interrupts in a system, each handling a specific function in the system: reading/writing to disk, calling device drivers, and so on.

Every time someone presses a key on a keyboard, the keyboard interrupt is called.

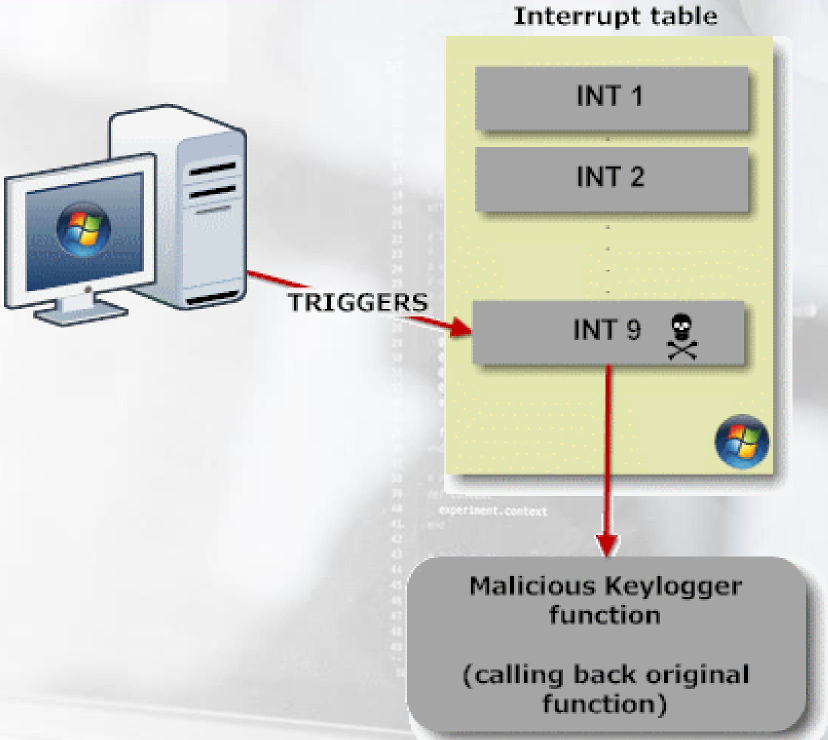
5.1.10.2 Rootkit Keylogger

The interrupt calls a particular function of the operating system that actually performs the operation intended for the key.

By taking control of this function, the rootkit manages to know which key has been pressed and records it for later use.

5.1.10.2 Rootkit Keylogger

First, the keylogger logs the key pressed and then it calls the original function of the operating system.



5.1.11 Bots

Bots are small pieces of software that get installed on millions of Internet-connected machines to perform Distributed Denial of Service or serving as spamming sources.

These Bots are commanded remotely by a so-called **Command and Control** server. The C&C server can instruct thousands or even millions of bots to perform a given operation simultaneously.

5.1.12 Ransomware

Ransomware is software that encrypts a computer or smartphone content with a secret key.

It then asks its victims for a ransom to give them the content back.

5.1.13 Data Stealing Malware

Data stealing malware has one precise goal: stealing the most important data on the victim's hard disk and sending it back to the attacker. Most of the time, this specific malware incarnation is **targeted** to a specific company and tailored to work on the target environment.

As an alternative, an attacker could use a backdoor to perform data stealing.

5.1.14 Worms

Worms spread over the network by exploiting operating systems and software vulnerabilities. Worms can also exploit default credentials or misconfigurations to attack a service or a machine.

Usually, worms are part of other malware, and they offer an entry point into the target system.

5.1.15 Video – Backdoor

Backdoor

In this video, you will see how to use different tools to create a backdoor on an exploited machine. Every backdoor has its own unique features, so you will learn how to choose the right tool for the job!

*Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.

5.1.16 References

If you want to study more on this subject, a book you can reference and some malware examples to test!

- Hacking Exposed Malware & Rootkits: Security Secrets and Solutions, Second Edition – Structure, detection and attack methods
- Malware samples – Do not run them on your computer! **Use a virtual machine as a testing environment!**

http://www.amazon.com/Hacking-Exposed-Malware-amp-Rootkits/dp/0071823077/ref=sr_1_3?s=books&ie=UTF8&qid=1423220013&sr=1-3

http://www.offensivecomputing.net/?q=taxonomy/term/1

Password Attacks

5.2 Password Attacks

How does this support my pentesting career?

Ability to:

- Gain persistent access to an exploited machine
- Choose the right method to attack passwords
- Exploit re-used credentials

5.2 Password Attacks

Passwords are the first and, most of the time, the only line of defense of systems, services and accounts against unauthorized access. To protect their users and applications, operating systems have to store passwords securely.

Passwords are usually stored inside files or databases. If they were **stored in clear-text**, attackers would just have to open the password file, or database, to steal them.

5.2 Password Attacks

Getting control over the credentials of an account means getting full control over the account and its privileges on a system.

Because of that, computer systems passwords must be stored in an **encrypted form**; this prevents a malicious local user from getting to know other users' passwords.

5.2 Password Attacks

To make things even harder for attackers, passwords are stored by using a **one-way encryption algorithm**; there is no way to know the password starting from its encrypted form.

Cryptographic hashing functions are used to transform a password from its clear-text form to an encrypted and safe to store form.

EXAMPLE

5.2 Password Attacks

When you log into your computer, you type your username and password. The operating system takes the password, hashes it and then tries to match the result against the saved hash in the password database. If the two values match, you log in successfully.

The operating system does not need to know the clear-text password!

5.2 Password Attacks

Password cracking is the process of recovering clear-text passwords starting from their hash.

It is basically a guessing process; the attacker tries to guess the password, hashes it and then compares the result against the password file.

5.2 Password Attacks

As you can imagine, trying to manually crack a password is more than impractical. To automate such processes, there are two main strategies:

- **Brute force attacks**
- **Dictionary attacks**

In the following slides, we will discuss the differences between the two, pros and cons, and the tools you can use to crack a password.

5.2.1 Brute Force Attacks

Q

*How can you be **certain** about finding someone's password?*

A

*You simply have to **try them all**!*

This is the method that brute force attacks use; they generate and test all the possible valid passwords, as this is the only method that gives you the certainty of finding a user's password.

5.2.1 Brute Force Attacks

In the following slides, we will discuss how *Brute Force* attacks work, their strengths, their weaknesses, and some tools that can be used to carry out a brute force attack.

5.2.1 Brute Force Attacks

Brute force attacks are the only way to be certain of finding someone's password. To automate a brute force attack, you have to write a program which generates every possible password.

Let's see how to implement a brute force algorithm!

5.2.1.1 Brute Force Algorithm

In the following slides, you will see how a brute force attack can be used to crack a password.

The algorithm proposed here works with a password of any length. It starts with one-character long passwords and increments the password size until a valid password is found.

5.2.1.1 Brute Force Algorithm

Here we see a pseudo-code version of a brute force password attack:

```
password_found = false
password_length = 1

while password_found == false
do
    while can_create_password_of_length(password_length)
    do
        password = create_password_of_length(password_length)
        if (hash(password) match attacked_hash)
        then
            password_found = true
    done
    password_length = password_length + 1
done
```

5.2.1.1 Brute Force Algorithm

To find an unknown password of unknown length, the algorithm must cycle over every possible lowercase character.

a → b → c → d → e → … → z

5.2.1.1 Brute Force Algorithm

It will then test uppercase characters.

a → b → c → d → e → ... → z

↓

A → B → C → D → E → ... → Z

5.2.1.1 Brute Force Algorithm

After cycling through uppercase characters, it will test numbers.

a → b → c → d → e → ... → z

A → B → C → D → E → ... → Z

0 → 1 → 2 → 3 → 4 → 5 → ... → 9

5.2.1.1 Brute Force Algorithm

Finally, it will cycle through special symbols.

a → b → c → d → e → … → z

A → B → C → D → E → … → Z

0 → 1 → 2 → 3 → 4 → 5 → … → 9

! → " → # → $ → % → & → … → @

5.2.1.1 Brute Force Algorithm

If the password is not found, the password length increases to two characters. The algorithm then chooses a value for the first character, performing the same operations we have just seen on the second one.

aa → ab → ... → a1 → ... → a! → ... → a@

5.2.1.1 Brute Force Algorithm

If no password is found, it changes the value of the first character and cycles again over the second one. This operation is performed for every possible value of the first character.

aa → ab → ... → ba → ... → z! → ... → @@

5.2.1.1 Brute Force Algorithm

The algorithm repeats.

Over lowercase characters...

a	a → b → c → d → e → ... → z
a	A → B → C → D → E → ... → Z
a	0 → 1 → 2 → 3 → 4 → 5 → ... → 9
a	! → " → # → $ → % → & → ... → @

5.2.1.1 Brute Force Algorithm

Uppercase characters...

J | a → b → c → d → e → ... → z

J | A → B → C → D → E → ... → Z

J | 0 → 1 → 2 → 3 → 4 → 5 → ... → 9

J | ! → " → # → $ → % → & → ... → @

5.2.1.1 Brute Force Algorithm

Numbers...

9	a → b → c → d → e → ... → z
9	A → B → C → D → E → ... → Z
9	0 → 1 → 2 → 3 → 4 → 5 → ... → 9
9	! → " → # → $ → % → & → ... → @

5.2.1.1 Brute Force Algorithm

and symbols.

=	a → b → c → d → e → ... → z
=	A → B → C → D → E → ... → Z
=	0 → 1 → 2 → 3 → 4 → 5 → ... → 9
=	! → " → # → $ → % → & → ... → @

5.2.1.1 Brute Force Algorithm

If the password is longer than two characters, then the algorithm moves to passwords that are three characters long, then four and so on until a valid password if found! Eventually, it will try every possible combination of letters, numbers, and symbols.

Given **enough time**, a brute force attack is **always successful**!

5.2.1.2 Brute Forcing Weaknesses

Brute force attacks appear to be the definitive weapon for password cracking because they always crack a password. In real-world attack scenarios, brute force attacks are used only when other attack vectors fail.

- Why does this happen?
- What are the intrinsic weaknesses of brute forcing?
- Why doesn't every attacker just use the brute force method when cracking a password?

5.2.1.2 Brute Forcing Weaknesses

Brute force attacks cycle through every possible input combination. They start from a single character and escalate to n-characters long, complex passwords. The resulting password could be made up of combinations of many letters, numbers, and symbols.

Generating and testing all those passwords takes time, even for a computer! In fact, one of the major caveats of a brute force attack is the time it takes to find the password.

5.2.1.2 Brute Forcing Weaknesses

If the password is poorly chosen by the user (short and made just of letters), then cracking it could take a couple of minutes or a couple of hours. If a password is long (fifteen or more characters) and made by upper and lower case letters, numbers and symbols, cracking it could take days, or even years.

Choosing a proper password, means making a brute force attack unfeasible because of timing constraints.

5.2.1.3 John the Ripper

Although writing a script to implement a brute force attack is not a difficult task per se, some tools include helpful functions such as session saving and multi-threading support, as well as the ability to brute force against different password file formats.

5.2.1.3 John the Ripper

John the Ripper is an extremely popular password cracking tool written for Unix-based operating systems. Nowadays, you can find sources and binaries for Linux, Mac OSX and Windows on Openwall website.

John the Ripper can mount both brute force and dictionary-based attacks against a password database.

http://www.openwall.com/john/

5.2.1.3 John the Ripper

You can use it on nearly one hundred encryption formats. You can check them out by using the `--list=formats` command line option:

```
# john --list=formats
des, bsdi, md5, bf, afs, lm, dynamic_n, bfegg, dmd5, dominosec, epi, hdaa,
ipb2, krb4, krb5, mschapv2, netlm, netlmv2, netntlm, netntlmv2, nethalflm,
md5ns, nt, phps, po, xsha, crc32, gost, keychain, lotus5, md4-gen, mediawiki,
mscash, mscash2, mskrb5, mssql, mssql05, mysql-sha1, mysql, nsldap, nt2, odf,
office, oracle11, oracle, osc, phpass, pix-md5, pkzip, racf, raw-md4,
raw-md5, raw-sha1, raw-sha1-linkedin, raw-md5u, salted-sha1, sapb, sapg,
sha1-gen, sip, vnc, wbb3, hmac-md5, hmac-sha1, raw-sha, raw-sha224,
raw-sha256, raw-sha384, raw-sha512, hmac-sha224, hmac-sha256, hmac-sha384,
hmac-sha512, xsha512, hmailserver, sybasease, dragonfly3-64, dragonfly4-64,
dragonfly3-32, dragonfly4-32, drupal7, sha256crypt, sha512crypt, episerver,
keepass, pwsafe, django, raw-sha1-ng, crypt, trip, ssh, pdf, wpapsk, rar,
zip, dummy
```

5.2.1.3 John the Ripper

The tool is extremely fast because of the high use of parallelization. It can also use many different cracking strategies during a brute force attack, and you can specify different password character sets, such as letters only or numbers only.

In the following slides, we will see how to use *John the Ripper* during a cracking session.

5.2.1.3 John the Ripper

In this scenario, after getting unauthorized access to a Linux machine, we made a copy of the password files:

- `/etc/passwd` that contains information about user accounts
- `/etc/shadow` that contains the actual password hashes

We want to use John the Ripper to crack some passwords.

5.2.1.3.1 Unshadow

John needs the username and the password hashes to be in the same file, so we need to use the *unshadow* utility. *Unshadow* comes with *John the Ripper*.

```
# unshadow passwd shadow > crackme
```

The actual cracking process can now start on the *crackme* file.

5.2.1.3.2 Brute force with John the Ripper

Usually, a password file contains passwords of multiple users. If you are interested in just cracking some of them, you can use the `–users` option.

To perform a pure brute force attack, you have to use the following syntax:

```
# john -incremental -users:<users list> <file to crack>
```

5.2.1.3.2 Brute force with John the Ripper

To brute force the password of the *victim* user, you have to type:

Example:

```
# john -incremental -users:victim crackme
```

5.2.1.3.2 Brute force with John the Ripper

Here you can see the John the Ripper output after finishing a simple cracking session:

```
# john -incremental -users:victim crackme
Created directory: /root/.john
Loaded 1 password hash (sha512crypt [64/64])
test                (victim)
guesses: 1  time: 0:00:00:28 DONE (Sun Feb 15 15:36:43 2015)  c/s: 856
trying: tomr - tina
Use the "--show" option to display all of the cracked passwords
reliably
```

5.2.1.3.2 Brute force with John the Ripper

To display the passwords recovered by *John*, you can use the `--show` option:

```
# john --show crackme
victim:test:1001:1003:,,,:/home/victim:/bin/bash
```

5.2.1.3.2 Brute force with John the Ripper

Most cracking sessions take hours or days to complete. During a long cracking session, you can press any key to show the status of *john*:

```
# john -incremental -users:victim crackme
Loaded 1 password hash (sha512crypt [64/64])
guesses: 0  time: 0:00:00:02 0.00%  c/s: 718  trying: 0101975 - 0100091
guesses: 0  time: 0:00:00:08 0.00%  c/s: 822  trying: andy21 - arc130
guesses: 0  time: 0:00:00:17 0.00%  c/s: 848  trying: chattest -
cheres96
```

5.2.1.3.2 Brute force with John the Ripper

Cracking a very long and complex password by brute force could take years, even on a very powerful machine. The time needed to crack a password is the only way to keep it safe from brute force attacks.

Keep this in mind when choosing a password or setting up a brute force attack!

5.2.2 Dictionary Attacks

Now that we have seen how brute force attacks work, we can study a generally faster attack: **dictionary attacks**.

Dictionary attacks do not try to generate every possible password, but they use a **dictionary of common passwords**, testing every single entry it contains.

5.2.2 Dictionary Attacks

Dictionary attacks are faster than pure brute force attacks because even a large dictionary, in order of magnitude, is smaller than the number of possible valid passwords for an account.

EXAMPLE

5.2.2 Dictionary Attacks

The number of commonly used 8-character long passwords is **3,051,366**, while the number of valid 8-character long lowercase passwords is **208,827,064,576.**

208 billion vs. 3 million passwords to test during an attack!!!

5.2.2.1 Performing a Dictionary Attack

To carry out a dictionary attack, you need:

- A password file containing the hashed passwords to crack.
- A dictionary, or wordlist, of passwords.
- A tool to test every password in the wordlist against the password file.

5.2.2.1 Performing a Dictionary Attack

During a dictionary attack, the password recovery tool used simply tests every entry in the wordlist.

Wordlists usually contain commonly used passwords such as "admin", "password1234" or "trustNO1".

5.2.2.2 Weaknesses of Dictionary Attacks

Poorly chosen or default passwords are more exposed to dictionary cracking.

Uncommon passwords could be safe, but only a truly random and long password can be considered safe.

EXAMPLE

5.2.2.2 Weaknesses of Dictionary Attacks

"`rXQqCI1\9_DhwX9RH\zF`" or "`rN9QmGAUC4WnMdUgUSsdVlghX`" are safe passwords, but they are not easy to remember, so you need a tool, such as Keepass, to securely store them. **Most users are lazy and do not do that!**

http://keepass.info/

5.2.2.2 Weaknesses of Dictionary Attacks

According to https://howsecureismypassword.net/, the previous passwords are also secure from pure brute force attacks.

A brute force attack against each of those passwords would take respectively:

- 35 sextillion ($35 * 10^{21}$ years) for the first one
- 5 octillion ($5 * 10^{27}$ years) for the second one

5.2.2.2 Weaknesses of Dictionary Attacks

However, this does not mean that only a strong password can resist a dictionary attack. It simply could not be included in the dictionary!

If the password is not in the wordlist, the tool performing a dictionary attack will not be able to crack the password.

5.2.2.3 Mangling words

Being unable to crack a password just because of case differences or some digits at the end of a word would be a shame. Because of that, cracking tools provide some options to **mangle** the words in a dictionary.

Example:

Some variations on "cat" could be: cat12, caT, CAT, Cat, CAt, c@t and so on...

5.2.2.4 Dictionary Attacks with John the Ripper

You can run dictionary attacks with *John* by passing it the `-wordlist` argument. You can also specify a custom wordlist.

```
$ john -wordlist<=custom wordlist file> <file to crack>
```

5.2.2.4 Dictionary Attacks with John the Ripper

You can also apply some mangling by using the `–rules` parameter:

```
$ john -wordlist<=custom wordlist file> -rules <file to crack>
```

EXAMPLE

5.2.2.4 Dictionary Attacks with John the Ripper

In this example, we want to crack the *crackme* file by using the *John* default wordlist:

```
# john -wordlist -users=victim,victim2 crackme
Loaded 2 password hashes with 2 different salts
(sha512crypt [64/64])
guesses: 0  time: 0:00:00:05 DONE (Mon Feb 16
17:28:00 2015)  c/s: 1415  trying: paagal - sss
```

5.2.2.4 Dictionary Attacks with John the Ripper

If the default wordlist does not work, you can use a custom one. We first check the content of the custom wordlist and then use it with *John*:

```
# cat mywordlist
mysteryguy
MEGAPASSWORD
# john -wordlist=mywordlist -users=victim,victim2 crackme
Loaded 2 password hashes with 2 different salts (sha512crypt [64/64])
mysteryguy       (victim)
guesses: 1  time: 0:00:00:00 DONE (Mon Feb 16 17:29:43 2015)  c/s: 25.00  trying:
mysteryguy - MEGAPASSWORD
Use the "--show" option to display all of the cracked passwords reliably
```

5.2.2.4 Dictionary Attacks with John the Ripper

As a last resort, you can enable dictionary mangling:

```
# cat mywordlist
mysteryguy
MEGAPASSWORD
# john -wordlist=mywordlist -rules -users=victim,victim2 crackme
Loaded 2 password hashes with 2 different salts (sha512crypt [64/64])
megapassword      (victim2)
mysteryguy        (victim)
guesses: 2  time: 0:00:00:00 DONE (Mon Feb 16 17:36:07 2015)  c/s: 512  trying:
mysteryguy - Megapassword7
Use the "--show" option to display all of the cracked passwords reliably
```

5.2.2.5 Installing Password dictionaries

You can find some useful password dictionaries as part of the OWASP Seclists Project on GITHub. If you use Kali Linux, you can install the `seclists` package.

```
# apt-get install seclists
```

After installing it, you will find the dictionaries in:
`/usr/share/seclists/Passwords/`

https://www.owasp.org/index.php/OWASP_SecLists_Project
https://github.com/danielmiessler/SecLists/tree/master/Passwords

5.2.3 Rainbow Tables

There is another very clever way to crack passwords: **rainbow tables.**

Rainbow tables offer a tradeoff between the processing time needed to calculate the hash of a password and the storage space needed to mount an attack.

5.2.3 Rainbow Tables

Discussing in detail the mathematical principles behind rainbow tables is out of the scope of this course.

To understand how the attack works you only need to know that a rainbow table is a table containing links between the results of a run of one hashing function and another.

5.2.3 Rainbow Tables

The space needed to store a rainbow table depends on how many characters are allowed in a password (upper and lower case characters, digits, symbols) and its length. Moreover, the specific hash function used to store the password plays a role.

Rainbow tables' sizes vary from hundreds of megabytes to hundreds of gigabytes.

5.2.3 Rainbow Tables

By performing a lookup in the rainbow tables cracking tools will save the computational time needed to hash every candidate password.

This approach reduces a cracking session time from days to seconds!

5.2.3.1 Rainbow Tables Limitations

The limit of rainbow tables is the storage space needed to guarantee successful cracking sessions.

Example:

> A rainbow table that guarantees a 96.8% success rate when cracking a password thats length ranges from 1 to 9 characters and hashed with *MD5* weights 864GB!

Furthermore, they can be a great resource to crack simple and complex short password (5 - 8 character long passwords).

5.2.3.2 Ophcrack

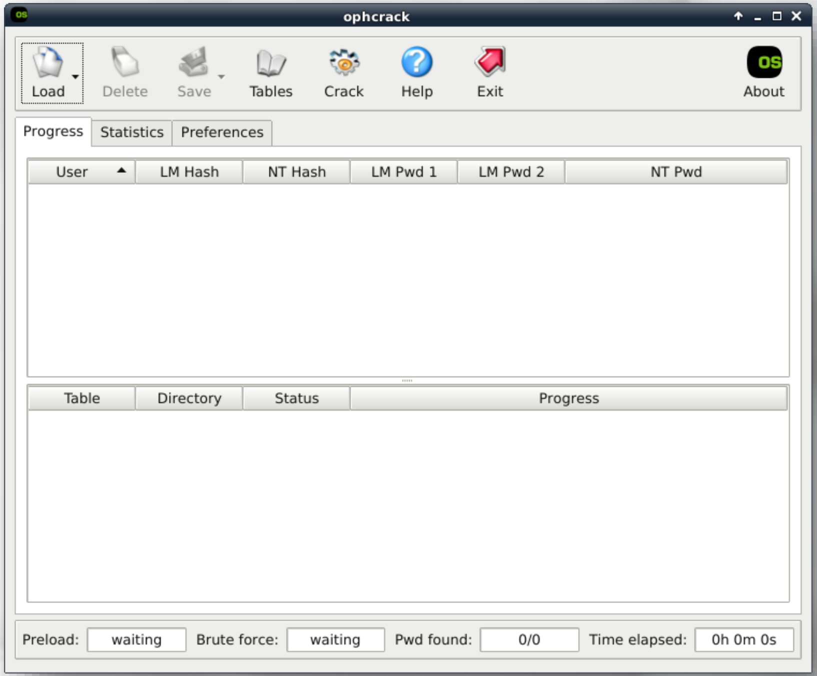
A great tool to perform rainbow cracking is ***ophcrack.*** It is a tool aimed at Windows password recovery, so you can use it only to crack Windows authentication passwords.

On its site, you will also find some **free rainbow tables**, where its size ranges can be from 300MB to 2TB.

5.2.3.2 Ophcrack

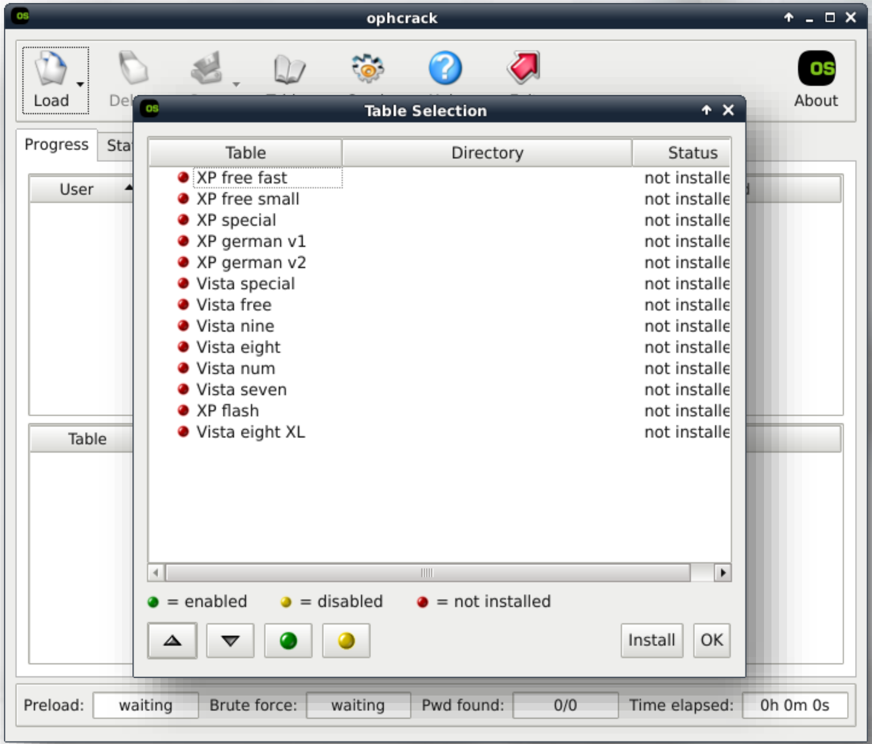
Ophcrack can run on Windows, Linux, Unix, and OSX.

Here we see the main window.



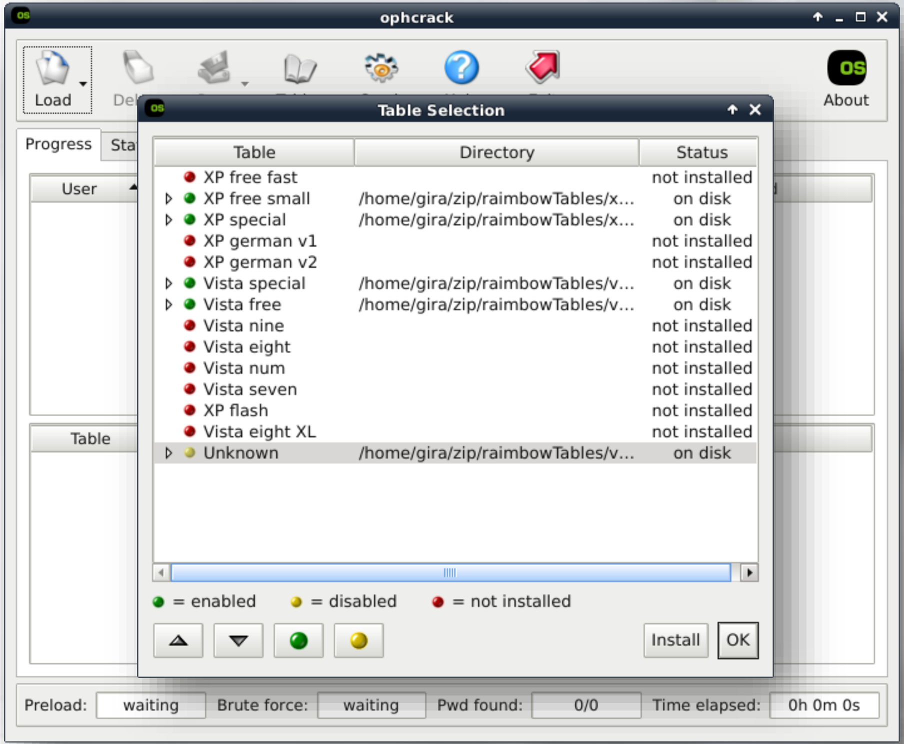
5.2.3.2 Ophcrack

To start cracking passwords, it is just a matter of clicking *Tables* in the main window and then selecting install.



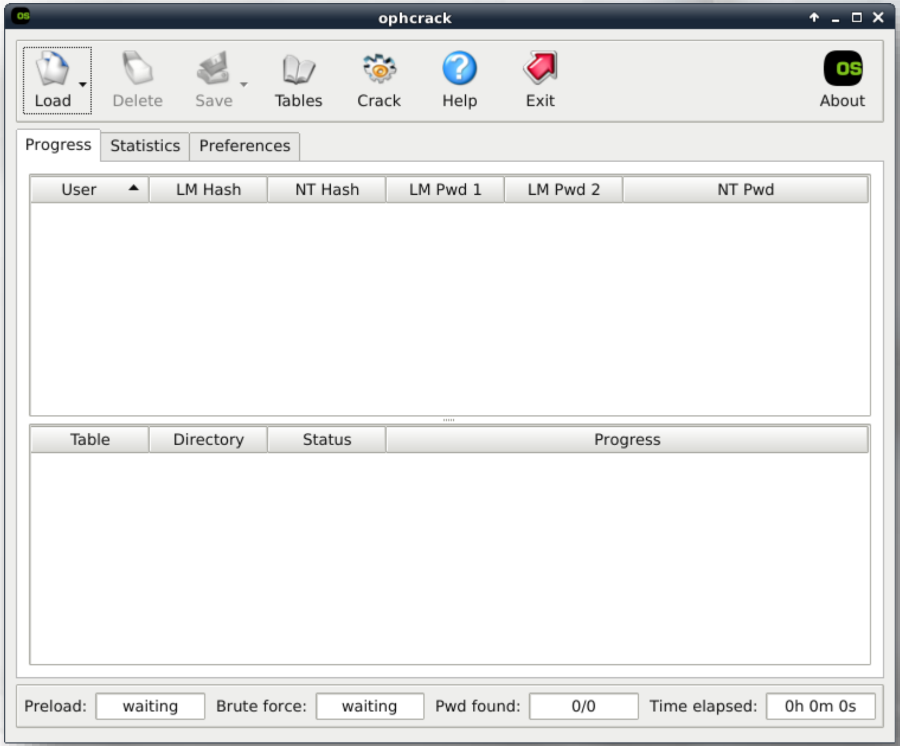
5.2.3.2 Ophcrack

You can install free or purchased tables and do not need to install them all.



5.2.3.2 Ophcrack

You can then load a password file by using the *Load* button in the main window.

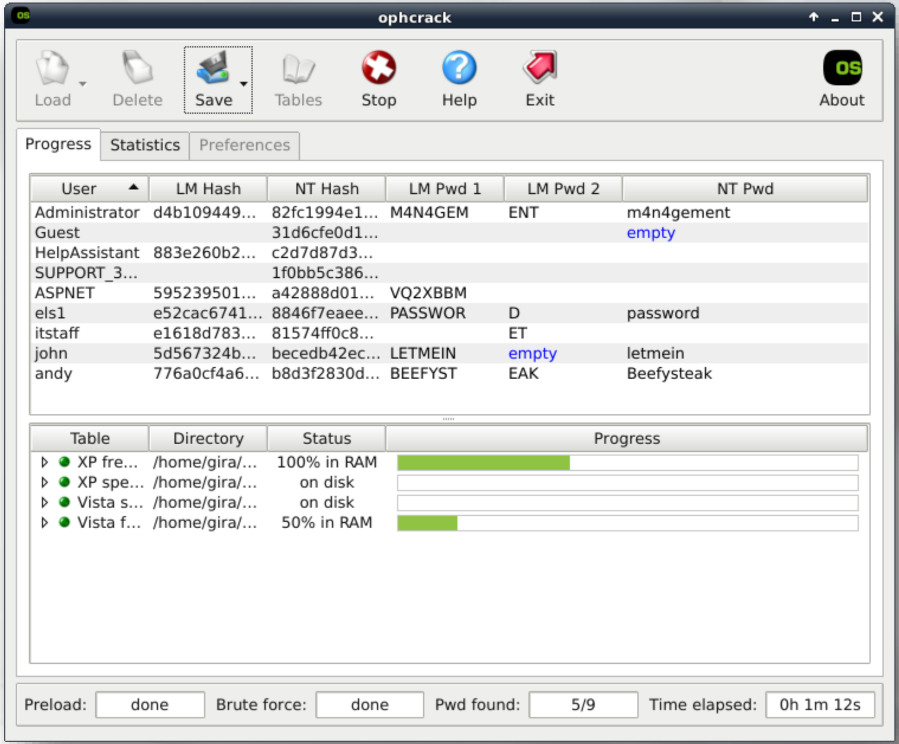


You will see how to get a password file from an exploited machine in the *Meterpreter* video at the end of this module.

5.2.3.2 Ophcrack

Finally, you click on the *Crack* button to start the process.

According to the tables you installed and the encryption format, you will be able to recover some or all the passwords!



5.2.4 Video – John the Ripper

John the Ripper

In this video, you will see how to use *unshadow* and *John the Ripper* to perform password cracking attacks against an exploited machine.

***Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.**

5.2.5 Hashcat Video

Hashcat

In this video, you will see how to use hashcat for password cracking using GPU on windows.

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

5.2.6 Conclusion

Password cracking is one of the ways to maintain access to a compromised machine. After getting access to a machine, a penetration tester can dump the password database and then crack it. This activity lets them access the machine by using valid credentials, thus being able to make their control over the machine persistent.

5.2.6 Conclusion

You can also apply the techniques you saw in this chapter to other password resources, like some credentials you dumped via SQL injection or a password file backup you somehow stole from a system.

5.3

Buffer Overflow Attacks

5.3 Buffer Overflow Attacks

How does this support my pentesting career?

- Remote code execution
- Privilege escalation attacks
- Understanding basics of memory corruption attacks.

5.3 Buffer Overflow Attacks

Many different attacks widely exploit **buffer overflow vulnerabilities**. They work by taking control of the execution flow of a piece of software or a routine of the operating system.

Taking control of the execution of a program means being able to force it to behave differently compared to what the application author designed.

5.3 Buffer Overflow Attacks

A buffer overflow attack can lead to:

- An application or operating system crash, thus causing a denial of service
- Privilege escalation
- Remote code execution
- Security features bypass

In the following slides, you will see how a buffer overflow vulnerability can arise and how a skilled attacker can exploit it.

5.3.1 Buffers

A **buffer** is an area in the computer Random Access Memory (RAM) reserved for temporary data storage. Data such as:

- User input
- Parts of a video file
- Server banners received by a client application
- And so on

5.3.1 Buffers

Buffers have a **finite size**; this means that they can only contain a certain amount of data.

Example:

If a client-server application is designed to accept only 8-characters long usernames, the username buffer will be 8 bytes long.

Other data in RAM

Other data in RAM

5.3.1 Buffers

If the developer of an application does not enforce buffers' limits, an attacker could find a way to write data beyond those limits, thus actually writing arbitrary code in the computer RAM; this can be exploited to get control over the program execution flow!

EXAMPLE

5.3.1.1 Buffer Overflow Example

A developer creates a text editor. By design the maximum length of a single line is 256 characters, so the editor does not accept user input that creates lines longer than 256 characters.

A penetration tester discovered that the application does not enforce this restriction when opening a file created with another editor. Moreover, when the editor opens a file, it inserts the first line in the line buffer.

5.3.1.1 Buffer Overflow Example

So if the penetration tester creates a file with just a single line made of, let’s say, 512 random characters and opens it with the editor, the application crashes! This means that the data in the file somewhat overwrote some of the editor code that is loaded in RAM.

The penetration tester then writes a script that first generates files with very long lines and then opens them with the application.

5.3.1.1 Buffer Overflow Example

After some trial and error, the penetration tester is able to generate files that, when opened with the editor, overwrite the program execution flow with valid code, giving the pentester control over the application!

This example should have given you a basic idea of how buffer overflow attacks work. Let’s delve into them more!

5.3.2 The Stack

Buffers are stored in a special data structure in the computer memory called a stack.

A **stack** is a data structure used to store data.

5.3.2 The Stack

You can imagine a stack as a pile of plates where you can add or remove just one plate at a time; this means that you can only add a plate on the top of the pile or remove a plate from the top of the pile.

5.3.2 The Stack

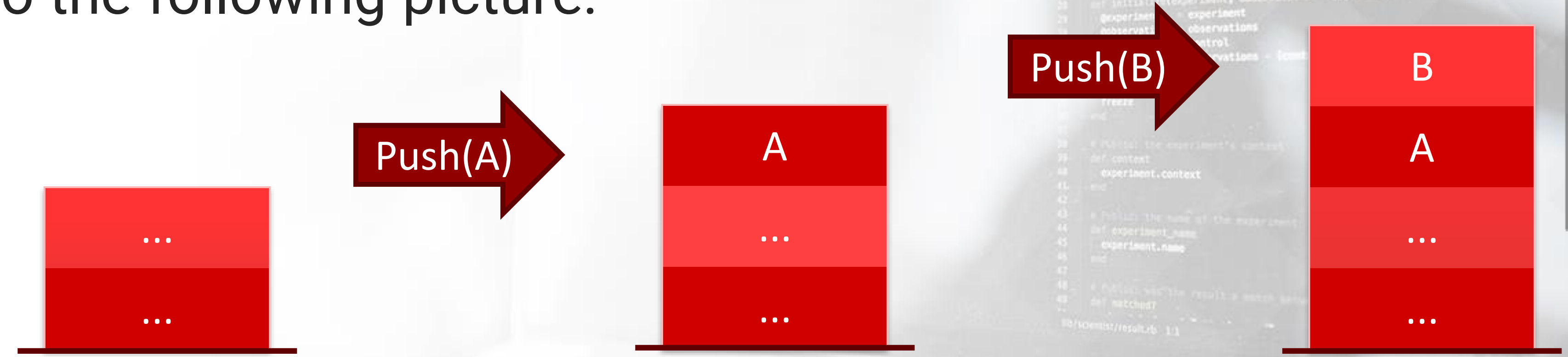
This approach is called **Last in First Out (LIFO)** and uses two methods:

- **Push**, which adds an element to the stack
- **Pop** removes the last inserted element

5.3.2.1 Push Operation

Push adds elements on the top of the stack.

If we add A and then B to the stack, it will change according to the following picture:



5.3.2.2 Pop Operation

Pop removes an element from the top of the stack. The element removed from the stack can be used by the software routine which popped it.

Example:

5.3.2.3 Allocating Space on the Stack

In modern operating systems, the stack is used in a more flexible way. Even if *push* and *pop* are still used, an application can randomly access a position on the stack to read and write data.

To save some stack space for later use, the application can simply reserve some memory allocations on the stack and then access them.

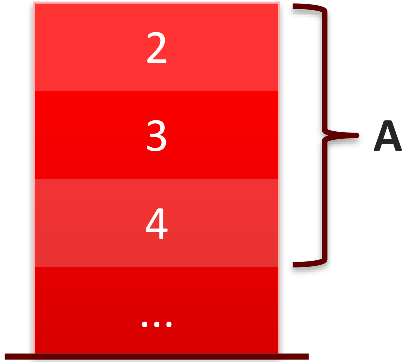
EXAMPLE

5.3.2.3 Allocating Space on the Stack

If an application needs to store an array of three integers, it can just allocate three positions on the stack.

The application code, in C, could be:

```
int A[3] = {2, 3, 4};
```

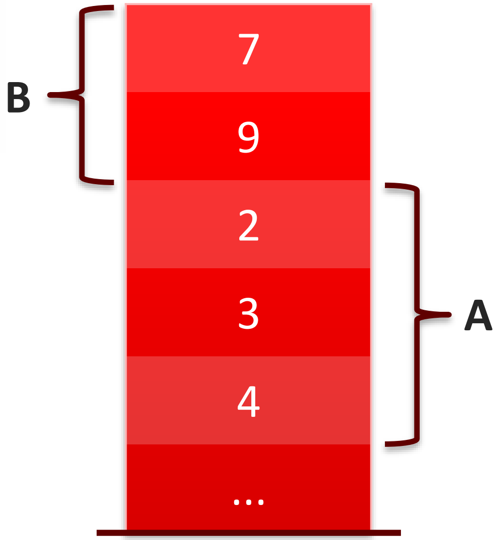


EXAMPLE

5.3.2.3 Allocating Space on the Stack

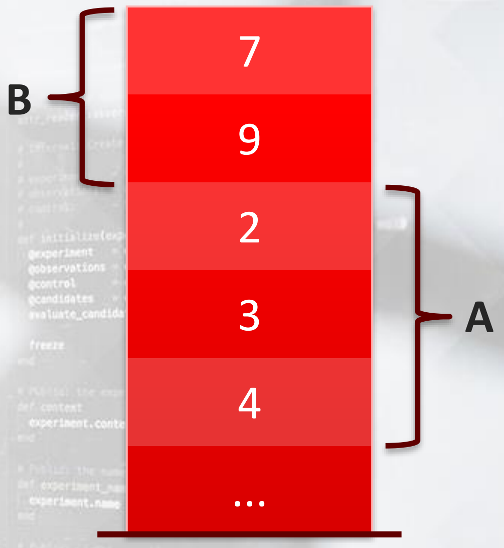
Allocating two arrays:

```
int A[3] = {2, 3, 4};
int B[2] = {7, 9};
```



5.3.2.4 Overflows in the Stack

The previous examples show how the stack grows bottom-up, while variables are written top-down. So, referring to the previous example, what happens if an attacker writes in *B* more than two integers?



They will be able to overwrite *A*!!!

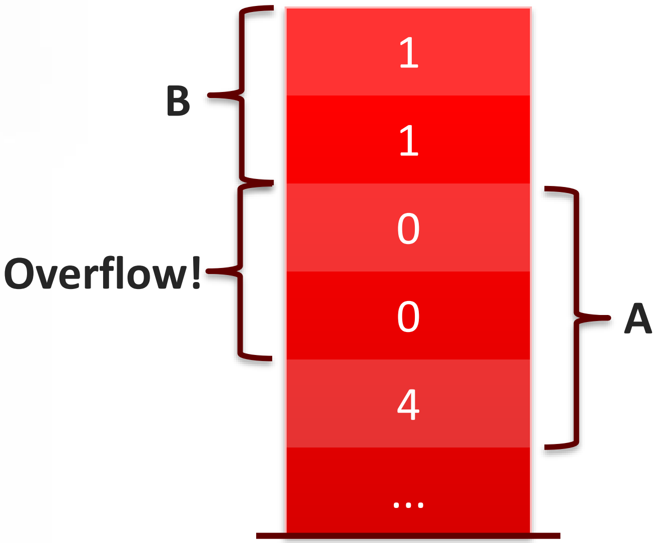
EXAMPLE

5.3.2.4 Overflows in the Stack

If an attacker finds a way to copy over B a **four-elements array**, they will overwrite a part of A.

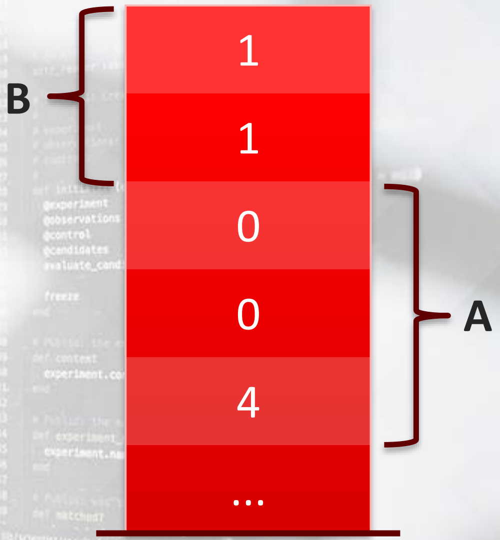
The picture shows an overflow performed by copying over *B* the following array:

```
int overflow[4] = {1, 1, 0, 0};
```



5.3.2.4 Overflows in the Stack

After the attacker exploited the buffer overflow vulnerability, the application will see *A* content as {0, 0, 4}!



5.3.3 The Stack in an Application

The stack used by applications and operating systems does not only contain data, but also **information about the execution flow**.

If we analyze a function call, we will see that the stack contains the function parameters, its local variables and the memory address where the execution of the program must continue after the function returns.

5.3.3 The Stack in an Application

So overwriting a function return address means getting control over the application. Moreover, if an attacker manages to write some valid code in RAM, they can force the victim function to run their code.

A raw overflow that just overwrites some memory locations will crash the application, while a well-engineered attack is able to execute code on the victim machine.

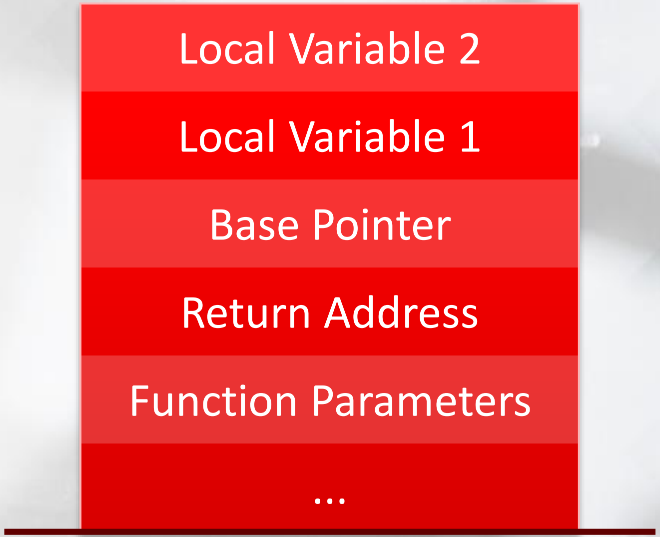
5.3.4 How Buffer Overflow Attacks Work

Deep analysis about how the stack works in an operating system is out of the scope of this course.

Local Variable 2
Local Variable 1
Base Pointer
Return Address
Function Parameters
…

5.3.4 How Buffer Overflow Attacks Work

Looking at this picture you should understand that if an attacker manages to overflow *Local Variable 1*, they are able to overwrite *Base Pointer* and then ***Return address!***



5.3.4 How Buffer Overflow Attacks Work

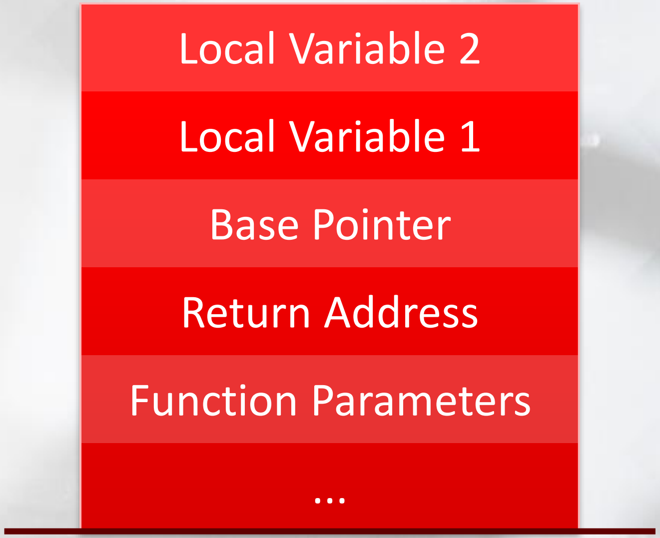
If they overwrite *Return Address* with the right value, they are able to control the execution flow of the program!

This technique can be exploited by writing custom tools and applications or by using hacking tools such as *Metasploit*.

Local Variable 2
Local Variable 1
Base Pointer
Return Address
Function Parameters
...

5.3.4 How Buffer Overflow Attacks Work

In the next module about network attacks, you will see a real-world buffer overflow exploitation with *Metasploit*!



5.3.4 How Buffer Overflow Attacks Work

Being able to write a buffer overflow exploit requires a deep understanding of assembly programming, how applications and operating systems works and some exotic programming skills.

If you are interested in that, you can check out the eLearnSecurity Penetration Testing Professional Course!

References

References

Hacking Exposed Malware & Rootkits, Second Edition

http://www.amazon.com/Hacking-Exposed-Malware-amp-Rootkits/dp/0071823077/ref=sr_1_3?s=books&ie=UTF8&qid=1423220013&sr=1-3

Keepass

http://keepass.info/

John the Ripper

http://www.openwall.com/john/

OWASP Seclists Passwords

https://github.com/danielmiessler/SecLists/tree/master/Passwords

References

Malware samples

http://www.offensivecomputing.net/?q=taxonomy/term/1

KeepassX

http://www.keepassx.org/

How Secure is my Password?

https://howsecureismypassword.net/

Ophcrack

http://ophcrack.sourceforge.net/

Videos

Backdoors

In this video, you will see how to use different tools to create a backdoor on an exploited machine. Every backdoor has its own unique features. So, you will learn how to choose the right tool for the job!

John the Ripper

In this video, you will see how to use *unshadow* and *John the Ripper* to perform password cracking attacks against an exploited machine.

***Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.**

Videos

Hashcat

In this video, you will see how to use hashcat for password cracking using GPU on windows.

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*