



Technical report

Operating Systems ST0257



This document by Juan Manuel Young Hoyos is for educational purposes only.
This document should not be printed or shared.

March 01 of 2022

Contents

1	General Research	2
1.1	Assignment	2
1.2	Conceptual Map	2
1.3	Timeline	2
2	Operating Systems Interruptions	4
2.1	What is an O.S. Interruption?	4
2.2	Buffer overflow	4
2.2.1	How can we avoid it?	6
2.3	Heap Memory Leak	6
3	Disclaimer	7
3.1	Confidentiality Statement	7
3.2	Contact info	7
3.3	Assesment overview	7
4	Antecedents	8
4.1	considerations	8
4.2	examples	8
5	Objectives	9
5.1	considerations	9
5.2	results	9
6	Vulnerability analysis	10
6.1	Initial Recognition	10
6.2	Improvement	10

1 General Research

The first activity is simple, the idea is just research about the history of the operating systems.

1.1 Assignment

- Make a conceptual map about operating systems.
- Make a timeline that reflects the history and generation of operating systems.

1.2 Conceptual Map

This is the conceptual map using draw.io:

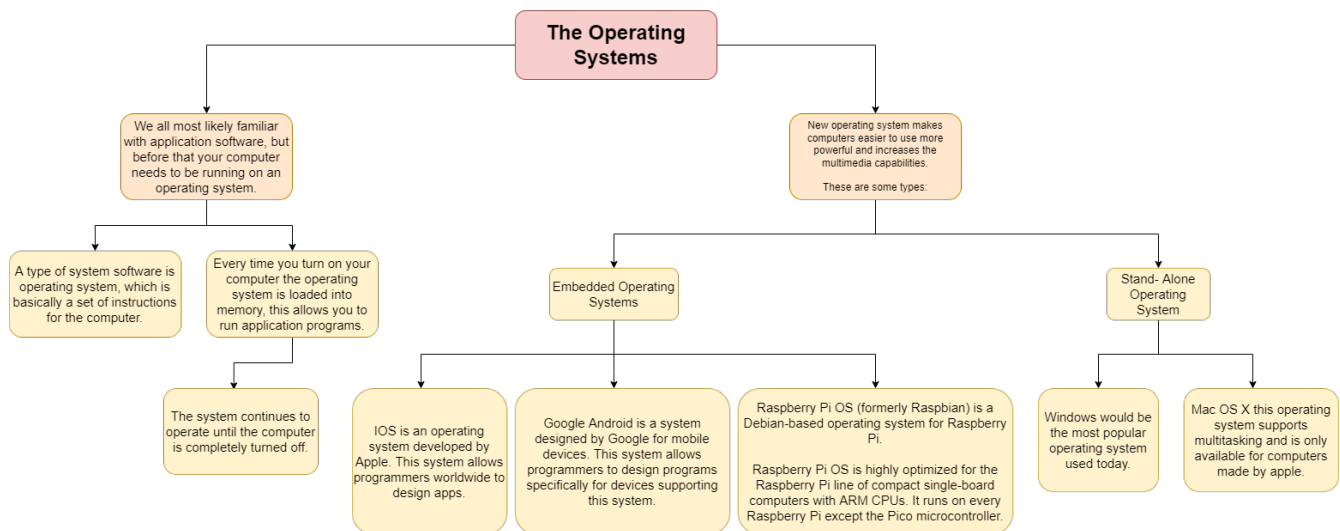


Figure 1: Operating Systems Conceptual Map

URL

[ConceptualMap.png](#) [GitHub file](#)

1.3 Timeline

This is the Operating System Timeline using draw.io:

URL

[OperatingSystemsTimeLine.png](#) [GitHub file](#)

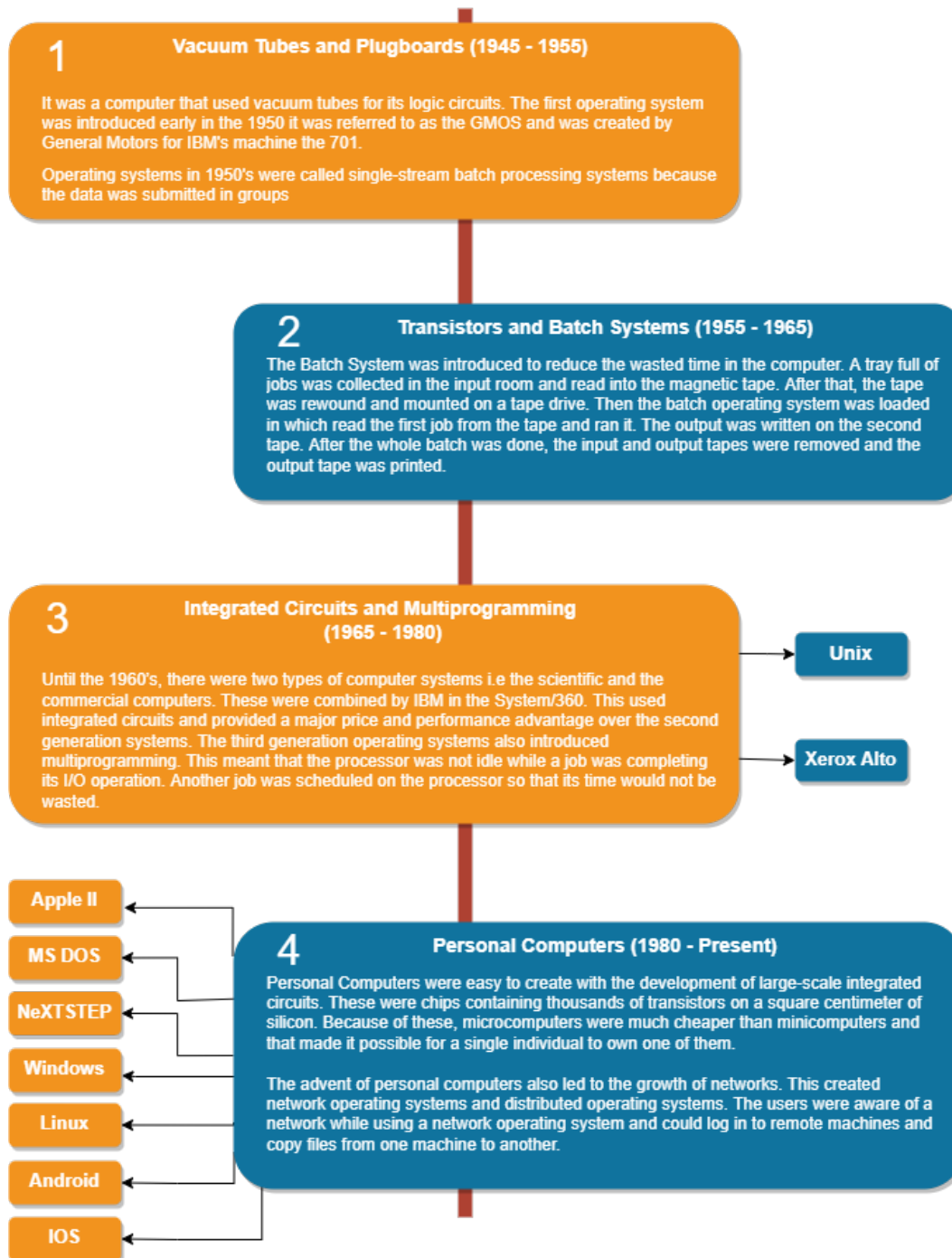


Figure 2: Operating Systems Timeline

2 Operating Systems Interruptions

The idea is just research and implementation of some of the operating system interruptions.

2.1 What is an O.S. Interruption?

An interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high-priority process requiring interruption of the current working process.

2.2 Buffer overflow

A buffer overflow occurs when a program or process attempts to write more data to a fixed-length block of memory, or buffer, than the buffer is allocated to hold. Buffers contain a defined amount of data; any extra data will overwrite data values in memory addresses adjacent to the destination buffer.

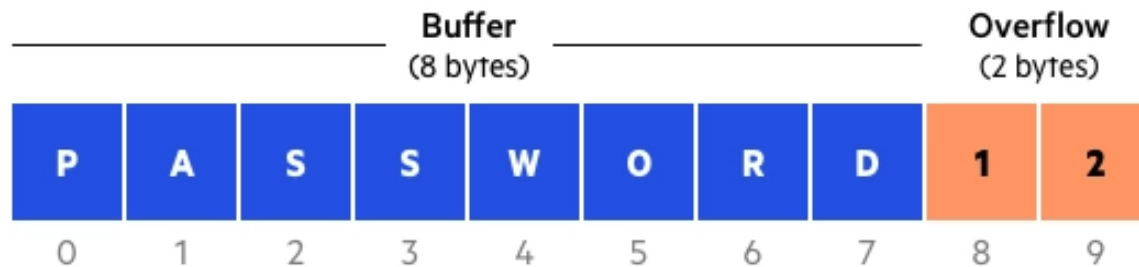


Figure 3: Buffer overflow diagram, image taken from imperva.com.

Now let's test it out with the language C and Rust. So we will have the following files for this test:

- ***Makefile***, this file allow us to build or clean the project.
- ***avoid memory leaks.rs***, this file shows an implementation of how Rust handles the memory leaks.
- ***buffer overflow.c***, this file shows an implementation of how to do a buffer overflow with C.
- ***get memory leaks.sh***, this script shows if Valgrind can help us with type of problems.

Now let's run the following *Makefile* with the command *make*, and we should get an executable.

```
1 # https://github.com/Youngermaster/ST0257-Operating-Systems/blob/main/Challenges/Challenge_1/
  BufferOverflow/Makefile
2
3 CC=gcc
4 CFLAGS=-g -Wall
5 EXE=buffer_overflow
6 REXE=avoid_memory_leaks
7
8 all: $(EXE)
9
10 %: %.c
11     $(CC) $(CFLAGS) $< -o $@
12
13 clean:
14     rm -rf *.o $(EXE)
15
16 rust:
17     rustc $(REXE).rs
18
19 cleanRust:
20     rm -rf *.o $(REXE)
```

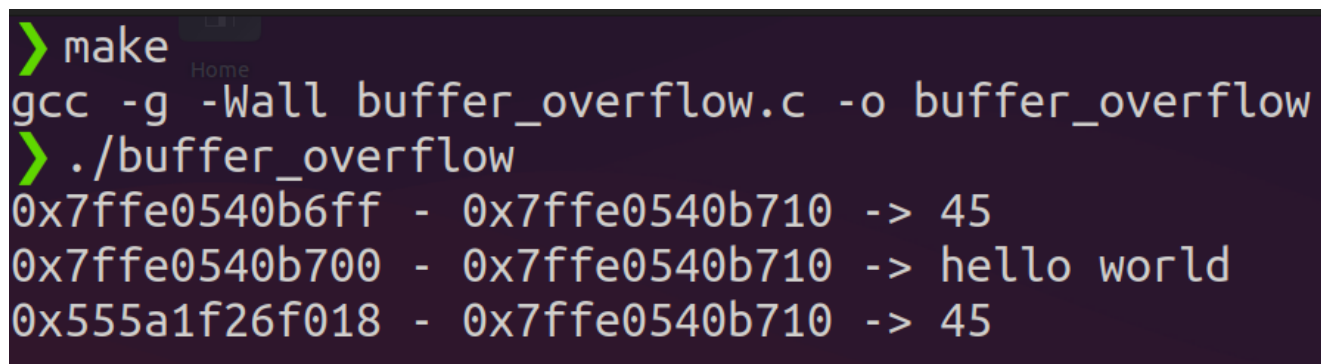
Code 1: This file allow us to build or clean the project.

The executable is made from the following code, where we are accessing to an outbound position.

```
1 // https://github.com/Youngermaster/ST0257-Operating-Systems/blob/main/Challenges/Challenge_1/
  BufferOverflow/buffer_overflow.c
2
3 #include <stdio.h>
4
5 int main(int argc, char const *argv[]) {
6     char *s = "hello world";
7     char c = s[20];
8     printf("%p - %p -> %d\n", &c, __builtin_frame_address(0), c);
9     printf("%p - %p -> %s\n", &s, __builtin_frame_address(0), s);
10    printf("%p - %p -> %d\n", &s[20], __builtin_frame_address(0), s[20]);
11    return 0;
12 }
```

Code 2: This file shows an implementation of how to do a buffer overflow with C.

Now, after the build we run it, and we get the following output:



```
> make
gcc -g -Wall buffer_overflow.c -o buffer_overflow
> ./buffer_overflow
0x7ffe0540b6ff - 0x7ffe0540b710 -> 45
0x7ffe0540b700 - 0x7ffe0540b710 -> hello world
0x555a1f26f018 - 0x7ffe0540b710 -> 45
```

Figure 4: Screenshot of the commands made in the terminal and the results.

We get the word "hello world", however, as we can see we got a random number, in this case the number 45. And what is the problem with that? The problem is that we are able to access to a random memory value, due to security issues and/or unexpected behaviours, imagine an airplane crash due to a little calculus modified by a Buffer Overflow.

2.2.1 How can we avoid it?

We have some options, but right now the most common are those two:

- We should pay attention to our C programs memory management.
- We can use another languages that solves that problem, using *Garbage Collectors*, *Borrowing and checkers*, etc.

Maybe a *Garbage Collector* approach is not a bad idea, however if we want maximum performance just maybe that is not the right path. Now let's check the *Borrowing and checkers* option with the language *Rust*. Now let's run the following code with the command `rustc ourRustFile.rs`

```
1 // https://github.com/Youngermaster/ST0257-0perating-Systems/blob/main/Challenges/Challenge_1/
  // BufferOverflow/avoid_memory_leaks.rs
2
3 fn main() {
4     // Fixed-size array (type signature is superfluous)
5     let xs: [i32; 5] = [1, 2, 3, 4, 5];
6
7     // Indexing starts at 0
8     println!("first element of the array: {}", xs[0]);
9
10    // ! Rust compilation will break this code immediately!
11    println!("first element of the array: {}", xs[20]);
12 }
```

Code 3: This file shows an implementation of how Rust handles the memory leaks.

And there is and interesting output, even if the *C* language allowed us to get a memory value far from our scope, *Rust* before compilation allow us to compile "*bad code*".

```
>rustc avoid_memory_leaks.rs
error: this operation will panic at runtime
--> avoid_memory_leaks.rs:9:48
9 |     println!("first element of the array: {}", xs[20]);
  |                                           ^^^^^^ index out of bounds: the length is 5 but the index is 20
= note: `[deny(unconditional_panic)]` on by default
error: aborting due to previous error
```

Figure 5: Screenshot of the commands made in the terminal and the results.

2.3 Heap Memory Leak

Memory leak occurs when programmers create a memory in heap and forget to delete it. The consequences of memory leak is that it reduces the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated and all or part of the system or device stops working correctly, the application fails, or the system slows down vastly.

3 Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflects the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. **EAFIT** prioritized the assessment to identify the weakest security controls an attacker would exploit. **EAFIT** recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

3.1 Confidentiality Statement

This document is the exclusive property of **EAFIT**. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of **EAFIT**.

3.2 Contact info

Contact info.			
Name	Title	Email	Contact
Juan	Cybersecurity Lead	juan@test	(99) 9999 9999
Manuel	Junior Pentester	manuel@test	(99) 9999 9999
Young	Junior Pentester	young@test	(99) 9999 9999

3.3 Assesment overview

From *Date 1* to *Date 2*, **EAFIT** engaged Penetration tests to evaluate the security posture of its infrastructure compared to current industry best practices. All testing performed is based on the NIST SP 800-115 Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide (v4), and customized testing frameworks.

Phases conducted for penetration testing are the following:

- Planning and preparation.
- Reconnaissance / Discovery.
- Vulnerability Enumeration / Analysis.
- Initial Exploitation.
- Expanding Foothold / Deeper Penetration.
- Cleanup.
- Report Generation.

4 Antecedents

The following document takes all the processes and results given by the audit made to the machine **Operating Systems ST0257** from the platform **HackTheBox**.



Figure 6: Details of the machine

URL

<https://app.hackthebox.com/machines/98>

4.1 considerations

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur molestie nulla nec hendrerit blandit. Aenean euismod tincidunt sapien at lacinia. Praesent at tortor nec nunc sodales posuere in eu mi. Fusce ac fringilla velit. Nam libero leo, consequat sit amet est varius, imperdiet iaculis massa. Suspendisse scelerisque eu erat eu cursus. Vivamus ac euismod ex. Integer euismod sem et euismod mattis. Integer vel rutrum velit, eu dictum nisi. Pellentesque venenatis et nisi a sodales. Integer pretium ut nisi at fringilla. Quisque nunc dui, consequat id turpis non, dignissim consectetur nunc.

4.2 examples

Morbi id turpis bibendum, ultrices turpis eu, molestie nunc. Duis aliquet aliquam turpis, vitae ultricies turpis. Ut tristique elementum nunc ac euismod. Proin viverra ultrices enim, et bibendum sem dignissim venenatis. Pellentesque non lectus nec erat congue viverra feugiat a urna. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nam vel ex sit amet nulla mollis lacinia. Nulla vehicula orci sit amet fermentum egestas. Mauris blandit ultricies sem, id convallis nunc malesuada non. Morbi venenatis ultricies leo, vel ullamcorper ligula mattis a.

5 Objectives

The idea is to check the machine state of the machine **Operating Systems ST0257**.

5.1 considerations

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur molestie nulla nec hendrerit blandit. Aenean euismod tincidunt sapien at lacinia. Praesent at tortor nec nunc sodales posuere in eu mi. Fusce ac fringilla velit. Nam libero leo, consequat sit amet est varius, imperdiet iaculis massa. Suspendisse scelerisque eu erat eu cursus. Vivamus ac euismod ex. Integer euismod sem et euismod mattis. Integer vel rutrum velit, eu dictum nisi. Pellentesque venenatis et nisi a sodales. Integer pretium ut nisi at fringilla. Quisque nunc dui, consequat id turpis non, dignissim consectetur nunc.

5.2 results

Morbi id turpis bibendum, ultrices turpis eu, molestie nunc. Duis aliquet aliquam turpis, vitae ultricies turpis. Ut tristique elementum nunc ac euismod. Proin viverra ultrices enim, et bibendum sem dignissim venenatis. Pellentesque non lectus nec erat congue viverra feugiat a urna. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nam vel ex sit amet nulla mollis lacinia. Nulla vehicula orci sit amet fermentum egestas. Mauris blandit ultricies sem, id convallis nunc malesuada non. Morbi venenatis ultricies leo, vel ullamcorper ligula mattis a.

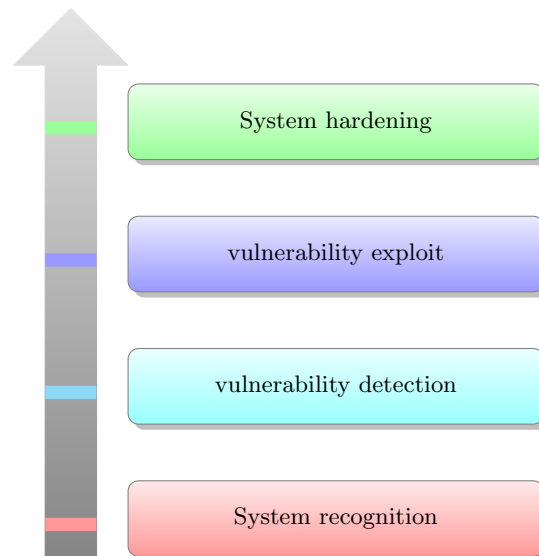


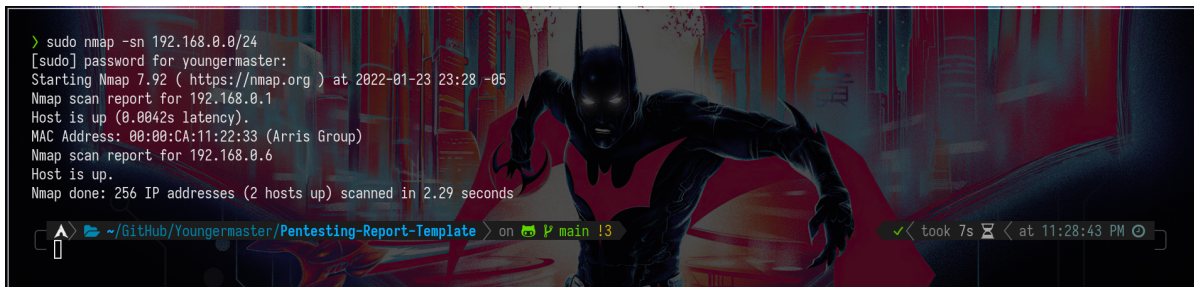
Figure 7: Workflow

6 Vulnerability analysis

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur molestie nulla nec hendrerit blandit. Aenean euismod tincidunt sapien at lacinia. Praesent at tortor nec nunc sodales posuere in eu mi.

6.1 Initial Recognition

Fusce ac fringilla velit. Nam libero leo, consequat sit amet est varius, imperdiet iaculis massa. Suspendisse scelerisque eu erat eu cursus. Vivamus ac euismod ex. Integer euismod sem et euismod mattis. Integer vel rutrum velit, eu dictum nisi. Pellentesque venenatis et nisi a sodales. Integer pretium ut nisi at fringilla. Quisque nunc dui, consequat id turpis non, dignissim consectetur nunc.



6.2 Improvement

Morbi id turpis bibendum, ultrices turpis eu, molestie nunc. Duis aliquet aliquam turpis, vitae ultricies turpis. Ut tristique elementum nunc ac euismod. Proin viverra ultrices enim, et bibendum sem dignissim venenatis. Pellentesque non lectus nec erat congue viverra feugiat a urna. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Nam vel ex sit amet nulla mollis lacinia. Nulla vehicula orci sit amet fermentum egestas. Mauris blandit ultricies sem, id convallis nunc malesuada non. Morbi venenatis ultricies leo, vel ullamcorper ligula mattis a.

```

1 # This function is taken from S4vitar's blog.
2 # https://s4vitar.github.io/bspwm-configuration-files/
3
4 ports="$(cat $1 | grep -oP '\d{1,5}/open' | awk '{print $1}' FS='/' | xargs | tr ' ' ',,')"
5 ip_address="$(cat $1 | grep -oP '\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}' | sort -u | head -n 1)"
6 echo -e "\n[*] Extracting information...\n" > extractPorts.tmp
7 echo -e "\t[*] IP Address: $ip_address" >> extractPorts.tmp
8 echo -e "\t[*] Open ports: $ports\n" >> extractPorts.tmp
9 echo $ports | tr -d '\n' | xclip -sel clip
10 echo -e "[*] Ports copied to clipboard\n" >> extractPorts.tmp
11 cat extractPorts.tmp; rm extractPorts.tmp

```

Code 4: This script allow us to extract nmap generated info

Donec ut tincidunt dolor. Curabitur sit amet porttitor magna, nec consectetur mi. Praesent quis congue tellus, a tincidunt mauris. Aenean sed luctus enim. Donec ut maximus nisi, sed malesuada erat. Aliquam sollicitudin ullamcorper sem vitae ultrices. Sed iaculis enim egestas, suscipit arcu ac, lacinia risus. Proin scelerisque mi eu feugiat euismod:

<i>TCP</i>
<i>Ports</i>
593, 1337

Vivamus vitae elit porta, tempor justo tincidunt, accumsan ligula. Proin nec magna sit amet leo dignissim sollicitudin sit amet ut quam:

```
> cat ../Hacking-Challenges/HackTheBox/0.common_utilities/nmap_port_scanner.sh

File: ../Hacking-Challenges/HackTheBox/0.common_utilities/nmap_port_scanner.sh

1  # -sCV gets the version and the services that runs on the given ports.
2  # -p$(ports) The given ports.
3  # -oN exports all the info in a "targeted" mode.
4
5  nmap -sCV -p$(ports) $ip -oN targeted
6
7  # Note: If the target blocks the pings, use the -Pn flag
8
9  nmap -sCV -p$(ports) $ip -oN targeted -Pn
```

~/GitHub/Youngermaster/Pentesting-Report-Template on main !4

Figure 8: These are the results of lorem ipsum

Ut vulputate fermentum scelerisque 8 and 11. Interdum et malesuada fames ac ante ipsum primis in faucibus.