



Adv. fuzzing reloaded ft. challenges

1º Parte



Antonio Morales

// QUIEN SOY

#define speakers

Antonio Morales

#define job

Security Researcher at  **GitHub**

#define twitter

@nosoyndiemas 

*using namespace **EkoParty**;*



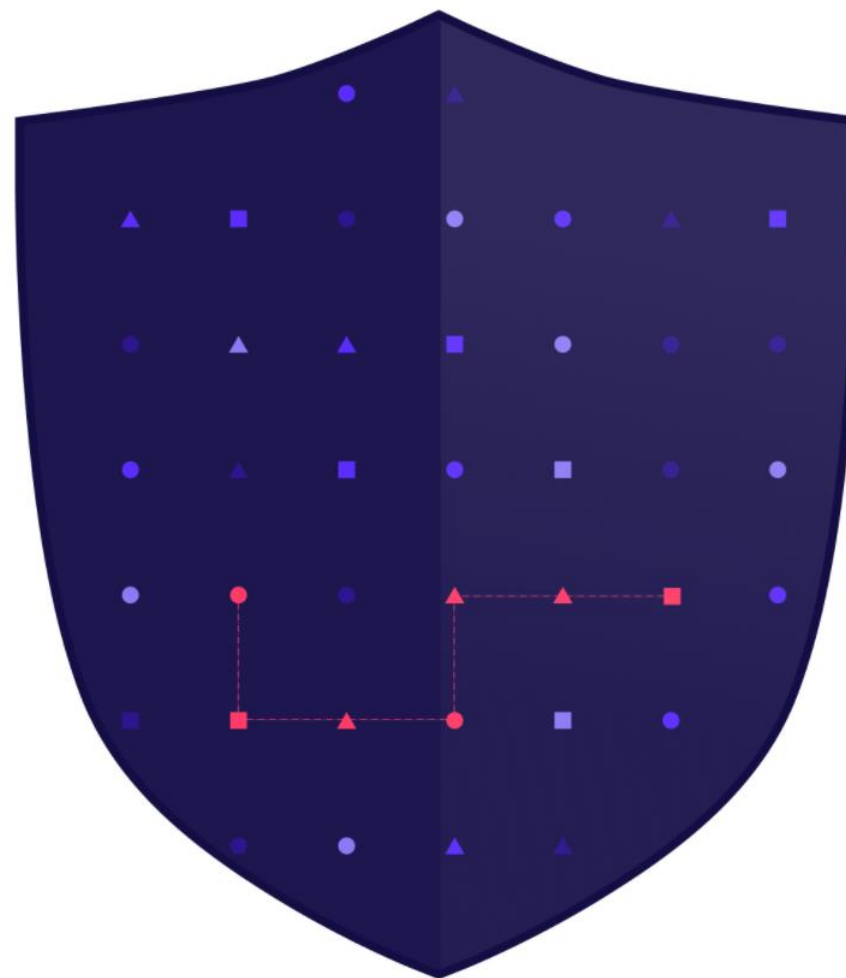
Security Lab



GitHub Security Lab

Securing the world's software, together

GitHub Security Lab's mission is to inspire and enable the community to secure the open source software we all depend on.

[Follow @GHSecurityLab](#)



Vulnerabilities we've disclosed

GitHub Security Lab researchers find vulnerabilities in key, widely-used open source projects. We then coordinate the disclosure of those vulnerabilities to security teams at those projects. We only publish vulnerabilities here after they've been announced by the affected projects' development teams and patches are available. See our [disclosure policy](#) below for more information.



September 9, 2021

GHSL-2021-123: ReDoS (Regular Expression Denial of Service) in Flask RESTX - CVE-2021-32838

ReDoS

Flask RESTX contains a regular expression that is vulnerable to ReDoS (Regular Expression Denial of Service).



Kevin Backhouse

September 9, 2021

GHSL-2021-108: ReDoS (Regular Expression Denial of Service) in



August 10, 2021

Don't shoot the emissary

[CodeQL](#) [Java](#) [CVE](#)

Check out how CodeQL detects some of the previously reported CVEs on NSA's Emissary by using its default rule set, how we were able to find an entirely new set of additional critical issues, and how the NSA leveraged GitHub code scanning and security advisories to ultimately address the issues.



Alvaro Munoz

August 5, 2021

Keeping your GitHub Actions and workflows secure Part 3: How to trust your building blocks

[Actions](#) [OpenSource](#) [Security](#)

In this article, we'll discuss sometimes less obvious attack vector — whose code GitHub Actions are running.



Jaroslav Lobacevski

July 13, 2021

Our shared common weaknesses

[Education](#) [CodeQL](#) [Advisories](#)

An overview of 2021's vulnerabilities so far.

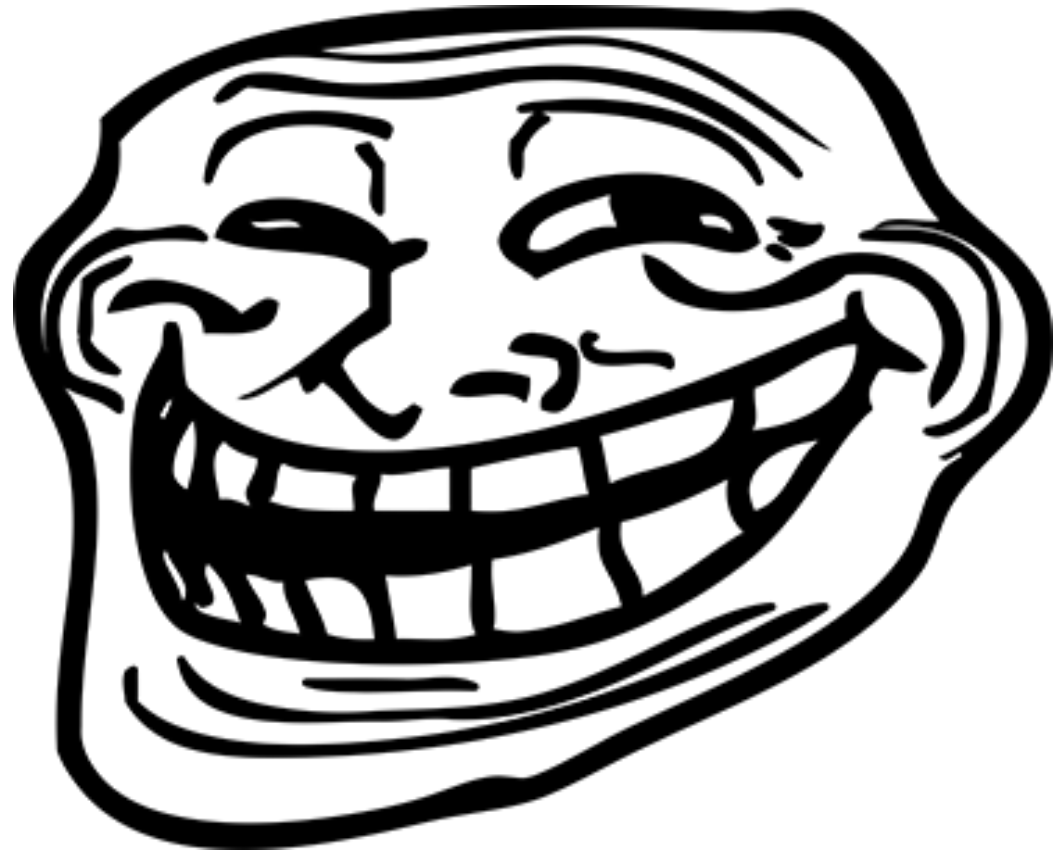


Jonathan Moroney

¿De qué vamos a hablar hoy?

¿Cómo hacernos ricos con BitCoin?





Temario del workshop

- El workshop estará dividida en 2 partes, de aproximadamente 2 horas cada una.
- Los horarios son los siguientes:
 - **Día 1:** 15 de Septiembre – 19:00h (ARG Time)
 - **Día 2:** 17 de Septiembre – 19:00h (ARG Time)
- En el **primer día**, realizaré una introducción acerca del fuzzing, partiendo desde 0 y mostrando paso a paso todo el procedimiento.
- En el **segundo día**, el formato será algo distinto. Se plantearán 2 retos a modo de CTF que los asistentes tendrán que resolver, guiados por las pistas que iré ofreciendo. Habrá **premios cortesía de GitHub** para los primeros en resolver los retos.

Temario del workshop

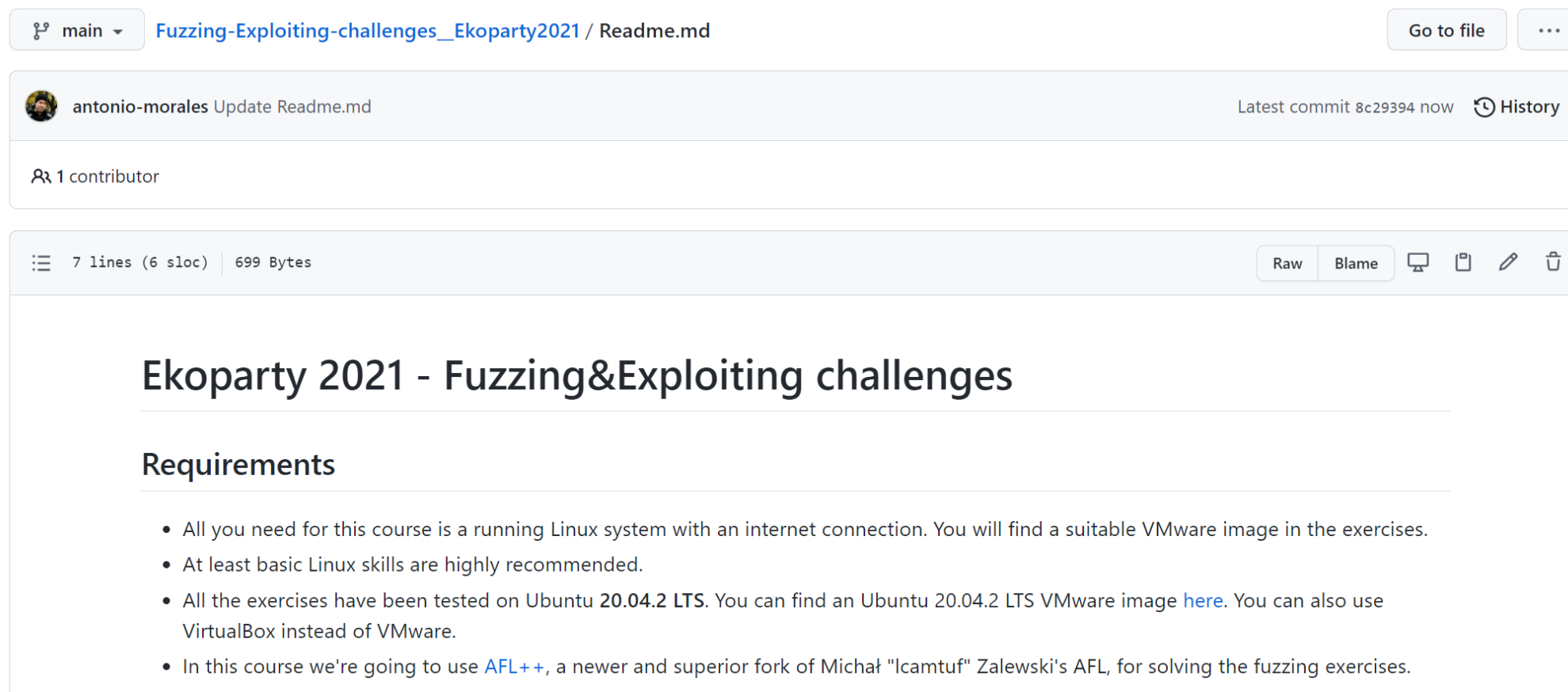
- Como novedad, este año incluiremos una pequeña **introducción al exploiting**, con 2 tipos distintos de vulnerabilidades:
 - Stack buffer overflow
 - Use-After-Free
- Al igual que con el fuzzing, realizaremos primero una introducción desde 0 de ambos tipos de vulnerabilidades, así como de su explotación.
- Posteriormente, en el segundo día habrá 2 retos distintos de exploiting, que los asistentes tendrán que tratar de resolver utilizando los conocimientos adquiridos.

Qué vamos a ver hoy

1. Introducción al fuzzing:
 1. ¿Qué es el fuzzing?
 2. Fuzzing guiado por cobertura
 3. Introducción a AFL++
2. Ejercicio de fuzzing sobre la librería LibRaw: demostración paso a paso de como encontrar una vulnerabilidad utilizando AFL++
 1. Address Sanitizer
3. Introducción a la explotación de vulnerabilidades de tipo stack buffer overflow

Repositorio para el workshop

Podrás encontrar todo lo necesario para el workshop en el siguiente enlace:
https://github.com/antonio-morales/Fuzzing-Exploiting-challenges_Ekoparty2021



The screenshot shows the GitHub interface for the repository 'Fuzzing-Exploiting-challenges_Ekoparty2021'. At the top, there's a navigation bar with a 'main' branch selector and a 'Go to file' button. Below this, a commit history bar shows 'antonio-morales' updated 'Readme.md' with the latest commit '8c29394' now. It also indicates '1 contributor'. The main content area shows the 'Readme.md' file with a header 'Ekoparty 2021 - Fuzzing&Exploiting challenges' and a section 'Requirements'. The requirements list four bullet points: 1. A running Linux system with internet connection and a suitable VMware image. 2. Basic Linux skills are recommended. 3. Exercises are tested on Ubuntu 20.04.2 LTS, with a link to find the image and a note about using VirtualBox. 4. The course will use AFL++, a fork of Michael 'lcamtuf' Zalewski's AFL.

main ▾ Fuzzing-Exploiting-challenges_Ekoparty2021 / Readme.md Go to file ...

antonio-morales Update Readme.md Latest commit 8c29394 now History

1 contributor

7 lines (6 sloc) 699 Bytes Raw Blame

Ekoparty 2021 - Fuzzing&Exploiting challenges

Requirements

- All you need for this course is a running Linux system with an internet connection. You will find a suitable VMware image in the exercises.
- At least basic Linux skills are highly recommended.
- All the exercises have been tested on Ubuntu 20.04.2 LTS. You can find an Ubuntu 20.04.2 LTS VMware image [here](#). You can also use VirtualBox instead of VMware.
- In this course we're going to use [AFL++](#), a newer and superior fork of Michał "lcamtuf" Zalewski's AFL, for solving the fuzzing exercises.

Introducción al fuzzing

Introducción al fuzzing

- Fuzz testing (o “fuzzing”) es una técnica de testeo de software automática.
- Se basa en enviar una gran cantidad de datos como entrada de un programa (mediante un programa de software), a la vez que se monitoriza la ejecución para detectar posibles excepciones/crashes.
- Estos datos pueden ser generados de forma aleatoria o en base a mutaciones de entradas

El fuzzer más simple

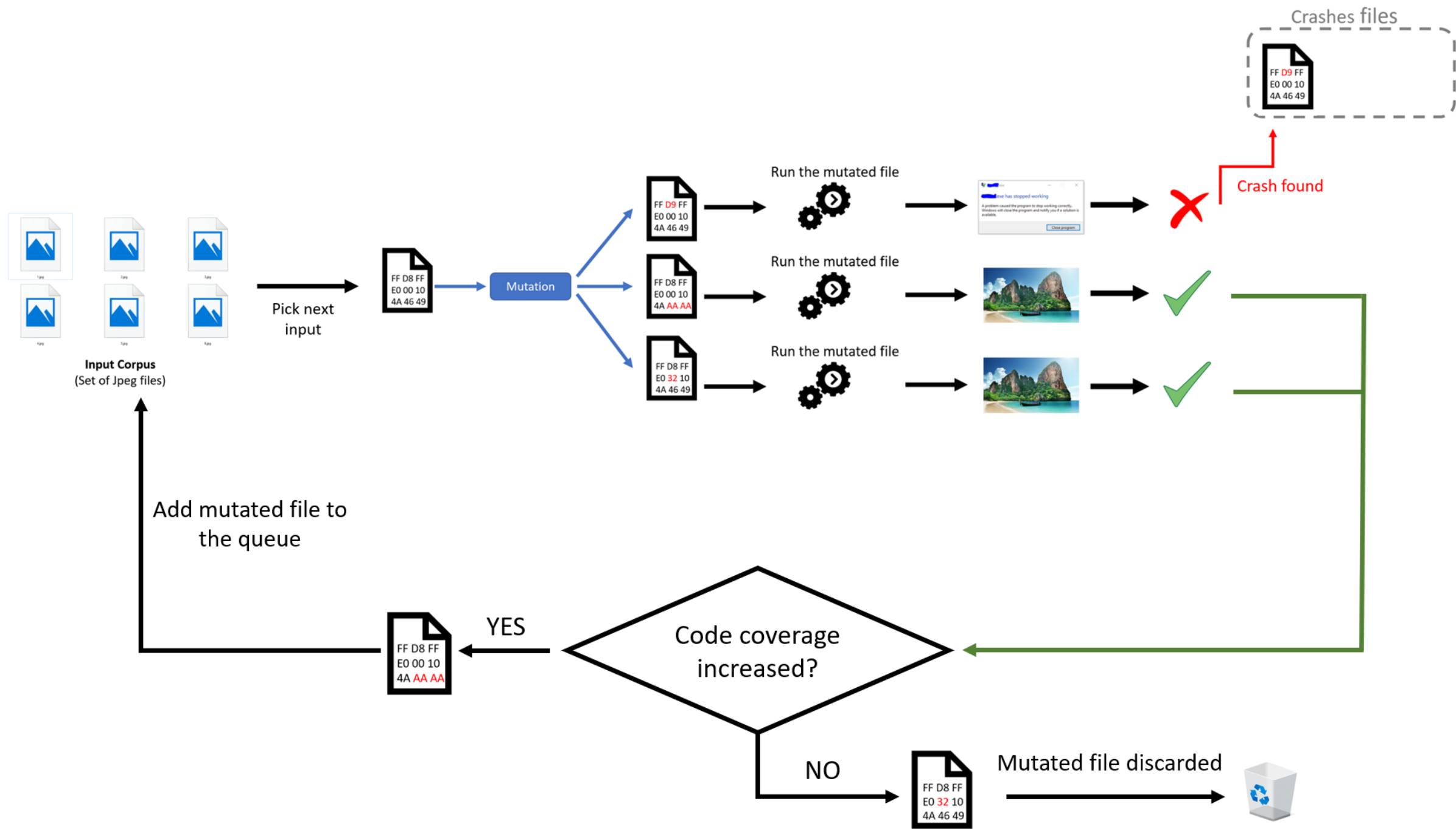
```
> cat /dev/urandom | nc -vv target port
```

Concepto 1: Fuzzing evolutivo

- Es una metaheurística inspirada en los algoritmos evolutivos, basada en la evolución y mutación a lo largo del tiempo de un subconjunto inicial
- Para dicha selección de candidatos más prometedores se utiliza un **criterio de selección** (por ejemplo la cobertura de código).

Concepto 2: Fuzzing guiado por cobertura

- Para incrementar las posibilidades de encontrar un nuevo crash, este tipo de fuzzers recolectan y comparan los datos de **cobertura de código** de los diferentes inputs.
- Se seleccionan y evolucionan aquellos inputs que incrementan los ratios de cobertura actual (nuevos paths de ejecución).
- Para poder obtener la cobertura de código, lo más común es “**instrumentar**” el programa a analizar. Esto se lleva a cabo (normalmente) durante el proceso de compilación (instrumentación de código).



Fuzzers evolutivos guiados por cobertura

- Dentro de esta categoría de fuzzers encontramos entre los más exitosos:
 - AFL
 - LibFuzzer
 - Honggfuzz
- Los 3 son ejemplos de fuzzer evolutivos guiados por cobertura

The background is dark with a pattern of wavy, horizontal lines in a slightly lighter shade. A faint, dark silhouette of a person is visible on the right side, appearing to be in a dynamic pose, possibly jumping or running.

PREGUNTAS??

DEMO TIME



Instalando AFL++

- El workshop estará dividida en 2 partes, de aproximadamente 2 horas cada una.
- Los horarios son los siguientes:
 - **Día 1:** 15 de Septiembre – 19:00h (ARG Time)
 - **Día 2:** 17 de Septiembre – 19:00h (ARG Time)
- En el **primer día**, realizaré una introducción acerca del fuzzing, partiendo desde 0 y mostrando paso a paso todo el procedimiento.

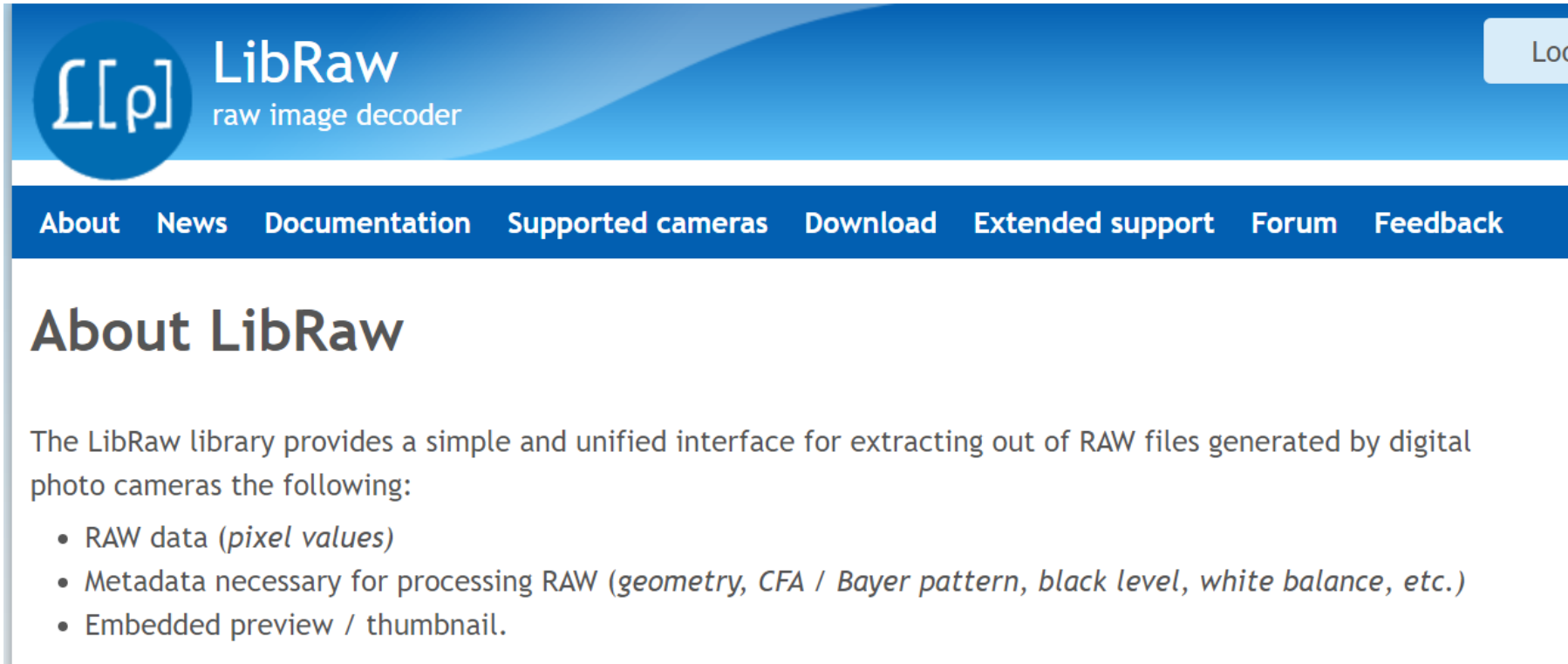
<https://github.com/antonio-morales/Fuzzing101/tree/main/Exercise%201#install-afl>



Fuzzing LibRaw for fun

LibRaw

- **LibRaw** es una libreria para el procesamiento de imagenes RAW generadas por distintas cámaras digitales
- Nuestro objetivo será encontrar una vulnerabilidad de tipo stack buffer overflow en **LibRaw 0.19.0**



The screenshot shows the top portion of the LibRaw website. At the top is a blue header bar with the LibRaw logo (a stylized 'L' and 'R' in a circle) and the text 'LibRaw raw image decoder'. To the right of the header is a 'Log' button. Below the header is a dark blue navigation bar with white links: 'About', 'News', 'Documentation', 'Supported cameras', 'Download', 'Extended support', 'Forum', and 'Feedback'. Below the navigation bar is the 'About LibRaw' section, which has a heading 'About LibRaw' and a paragraph: 'The LibRaw library provides a simple and unified interface for extracting out of RAW files generated by digital photo cameras the following:'. Below this paragraph is a bulleted list of features: 'RAW data (pixel values)', 'Metadata necessary for processing RAW (geometry, CFA / Bayer pattern, black level, white balance, etc.)', and 'Embedded preview / thumbnail.'

LibRaw
raw image decoder

Log

About News Documentation Supported cameras Download Extended support Forum Feedback

About LibRaw

The LibRaw library provides a simple and unified interface for extracting out of RAW files generated by digital photo cameras the following:

- RAW data (*pixel values*)
- Metadata necessary for processing RAW (*geometry, CFA / Bayer pattern, black level, white balance, etc.*)
- Embedded preview / thumbnail.

DEMO TIME



Stack-buffer overflow attacks

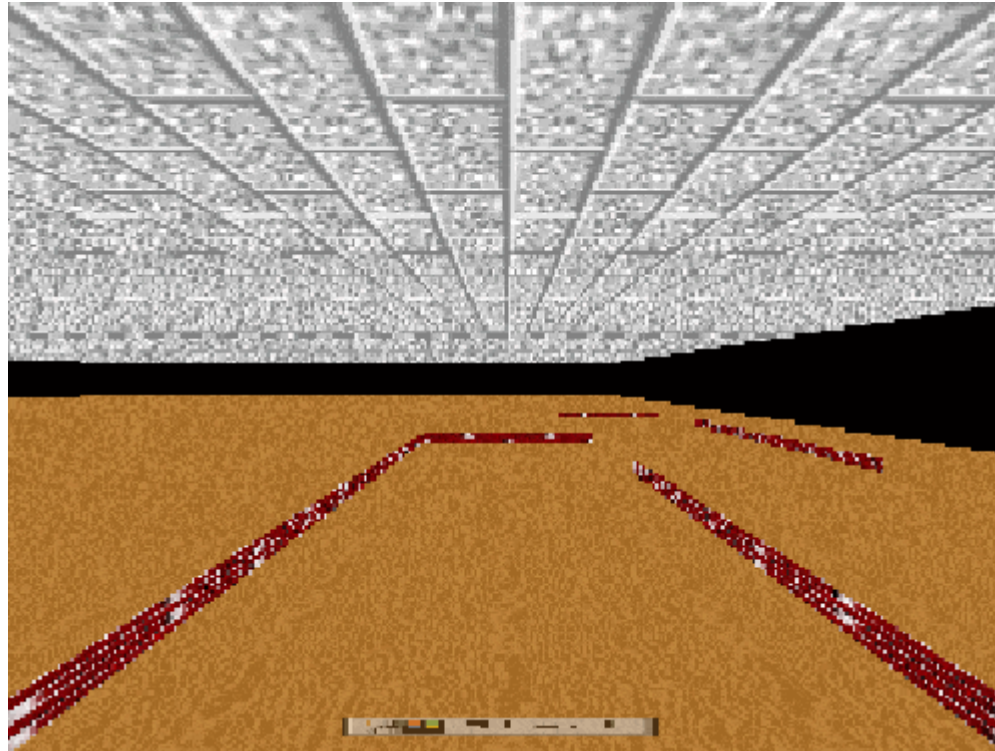
Buffer overflow – La teoría

“Un desbordamiento de buffer es un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copia sobre un área de memoria reservada a tal efecto (“buffer”).

Si dicha cantidad es superior a la capacidad preasignada, los bytes sobrantes se almacenaran en zonas de memoria adyacentes, sobrescribiendo su contenido original”

- LES HATTON -

Buffer overflow – En la práctica



Buffer overflow – En la práctica

```
/* Demostración de desbordamiento de buffer */  
  
#include <stdio.h>  
#include <string.h>  
  
int main(int argc, char *argv[])  
{  
    char buffer[10];  
  
    printf("Introduce una cadena: ");  
    scanf("%s", &buffer);  
  
    printf("Cadena Introducida:  %s \n", buffer);  
  
    return 0;  
}
```

Buffer overflow – En la práctica

```
C:\ Símbolo del sistema

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 12345
Cadena Introducida: 12345

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 1234567890
Cadena Introducida: 1234567890

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 123456789012345
Cadena Introducida: 123456789012345

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 12345678901234567890
Cadena Introducida: 12345678901234567890

C:\Documents and Settings\UWXP_SP1>
```

Buffer overflow – En la práctica

```
/* Demostración de desbordamiento de buffer */
```

```
#include <stdio.h>
#include <string.h>
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    char buffer[10];
```

```
    printf("Introduce una cadena: ");
```

```
    scanf("%s", &buffer);
```

```
    printf("Cadena Introducida:  %s \n", buffer);
```

```
    return 0;
```

```
}
```

Tamaño = **10**



Buffer overflow – En la práctica

```
C:\ Simbolo del sistema

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 12345
Cadena Introducida: 12345

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 1234567890
Cadena Introducida: 1234567890

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 123456789012345
Cadena Introducida: 123456789012345

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 12345678901234567890
Cadena Introducida: 12345678901234567890

C:\Documents and Settings\UWXP_SP1>
```

Ningún error?

¿ 20 ?

Buffer overflow – En la práctica

¿Y si probamos con 25 caracteres?

```
1 // DEMOSTRACION DE DESBOORDAMIENTO DE BUFFER
2
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(int argc, char *argv[])
7 {
```

B01.exe

B01.exe ha detectado un problema y debe cerrarse.

Si está en pleno proceso, puede perderse la información con la que esté trabajando.

Informe a Microsoft de este problema.

Se ha creado un informe de errores que puede enviarnos. Lo consideraremos como confidencial y anónimo.

Para ver los datos que contiene este informe de errores, [haga clic aquí](#).

C:\ Símbolo del sistema - B01.exe

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 12345
Cadena Introducida: 12345

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 1234567890
Cadena Introducida: 1234567890

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 123456789012345
Cadena Introducida: 123456789012345

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 12345678901234567890
Cadena Introducida: 12345678901234567890

C:\Documents and Settings\UWXP_SP1>B01.exe
Introduce una cadena: 1234567890123456789012345
Cadena Introducida: 1234567890123456789012345

Buffer overflow – En la práctica

Preguntas sin resolver:

- ¿Si nuestro buffer tenía un tamaño de 10 bytes, dónde se ha escrito el resto de la información que hemos introducido?
- ¿Por qué con 25 bytes nuestro programa ha producido un error y con 20 bytes no?

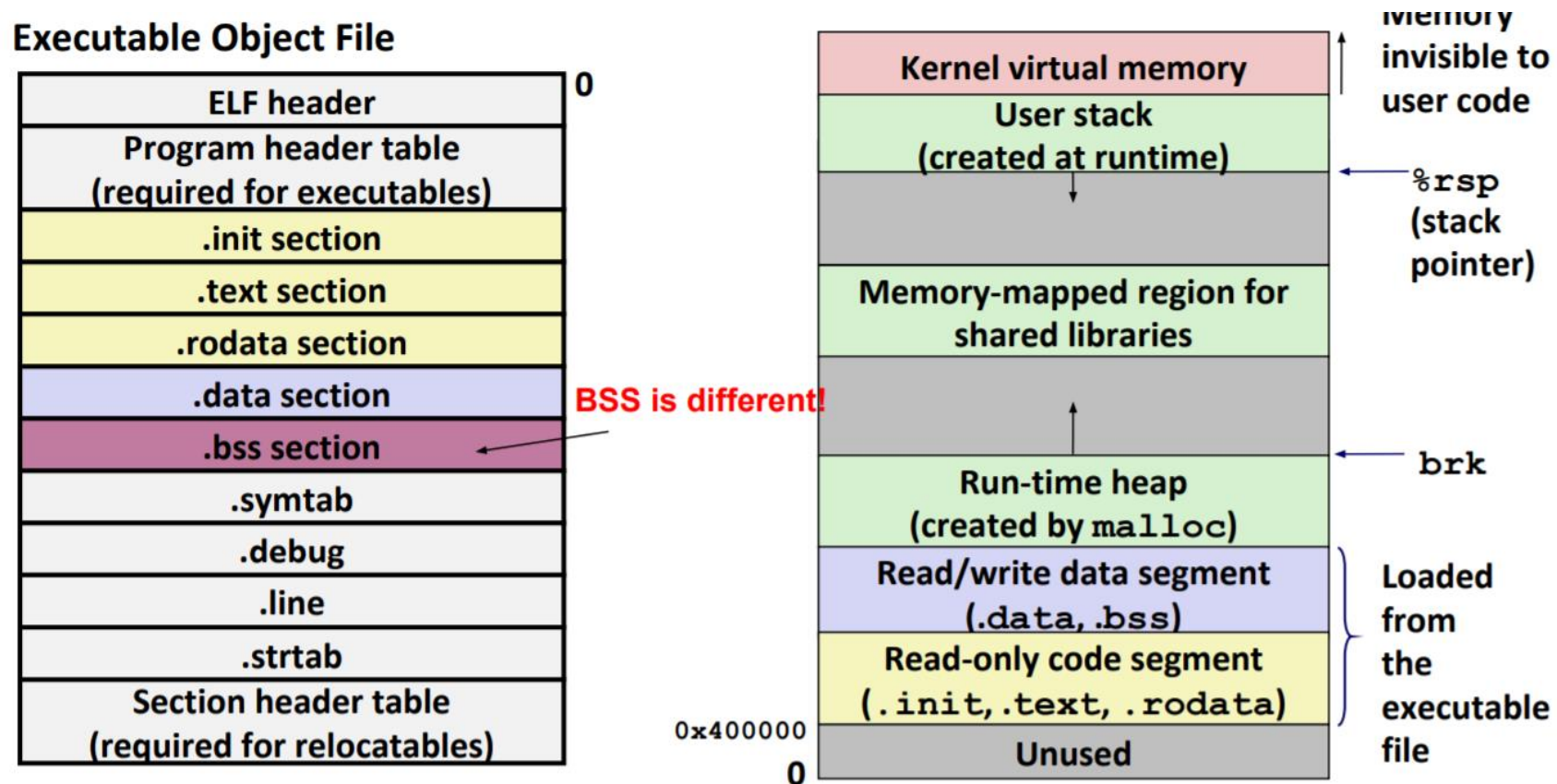
Un poco de teoría: El formato ELF

ELF header
Segment header table (required for executables)
.text section
.rodata section
.data section
.bss section
.symtab section
.rel.txt section
.rel.data section
.debug section
Section header table

- Un archivo ejecutable de Linux está estructurado de la siguiente forma.
- Dicho formato se conoce como ELF (Executable and Linkable Format), y es el formato estándar para los archivos ejecutables, el código objeto y las librerías compartidas en los sistemas UNIX.

Un poco de teoría: El formato ELF

- Es el sistema operativo el que se encarga de leer cada uno de los campos y mapearlos en memoria:



Buffer overflow – En la práctica

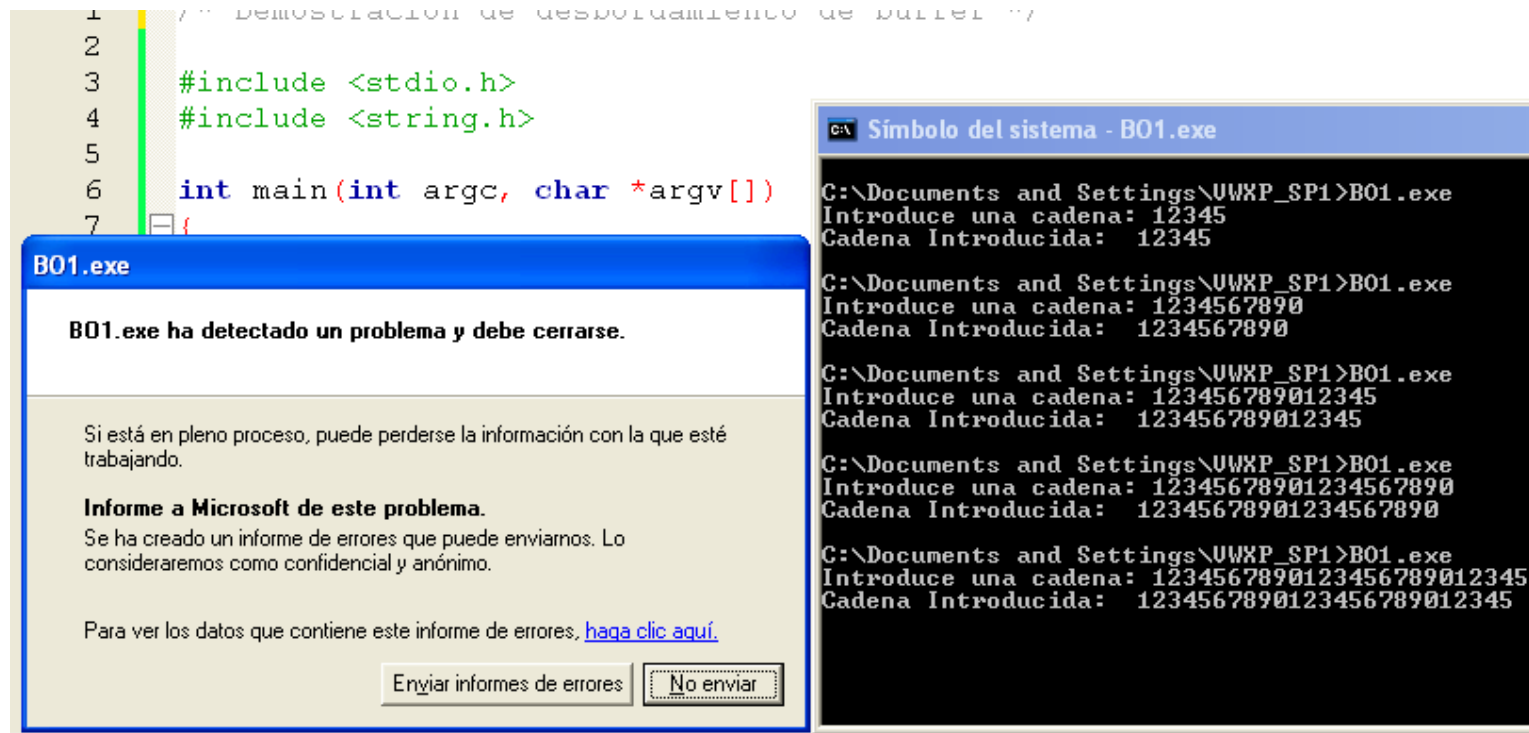
A modo de resumen, podemos mencionar las siguientes secciones:

- **.text/.code** : Contienen el código ejecutable (instrucciones máquina).
- **.rodata**: Contiene aquellos datos de solo lectura, como literales y constantes.
- **.data**: variables globales de la aplicación (datos inicializados)

Buffer overflow – En la práctica

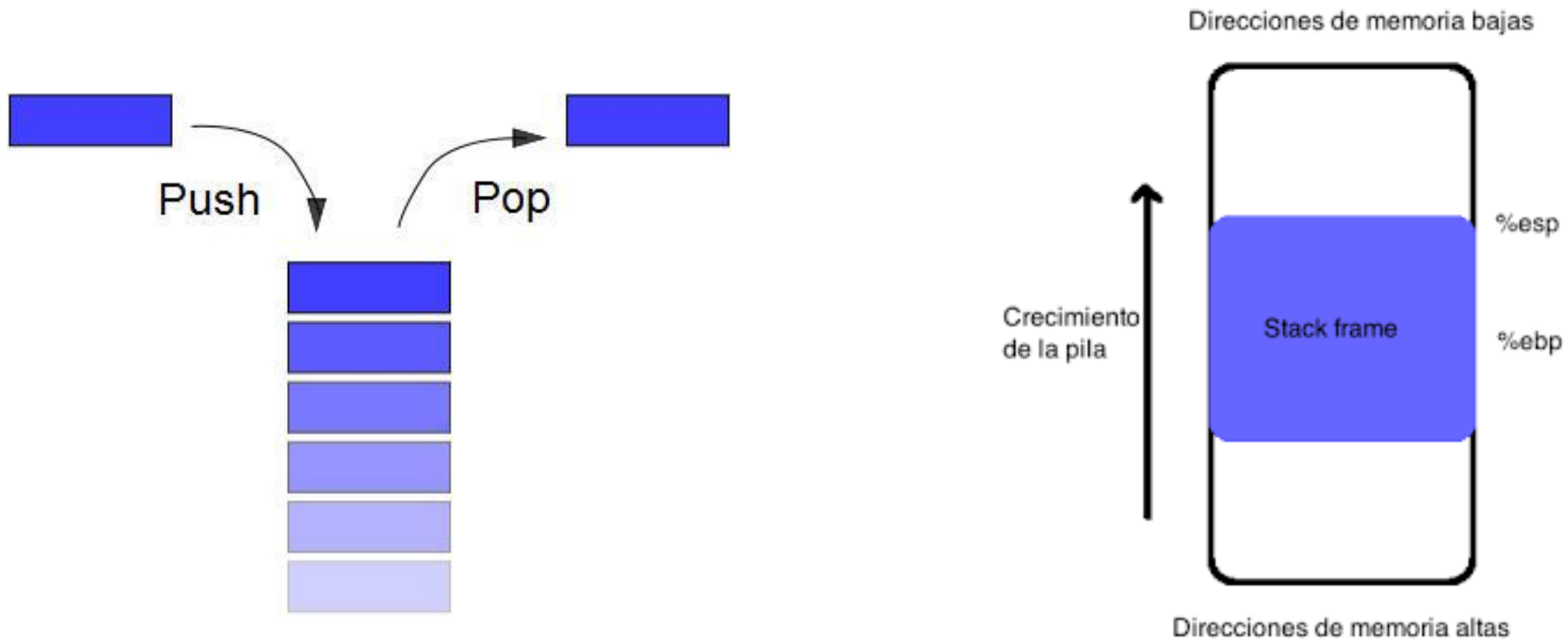
Preguntas sin resolver:

- ¿En qué sección se encuentra nuestro buffer desbordado?



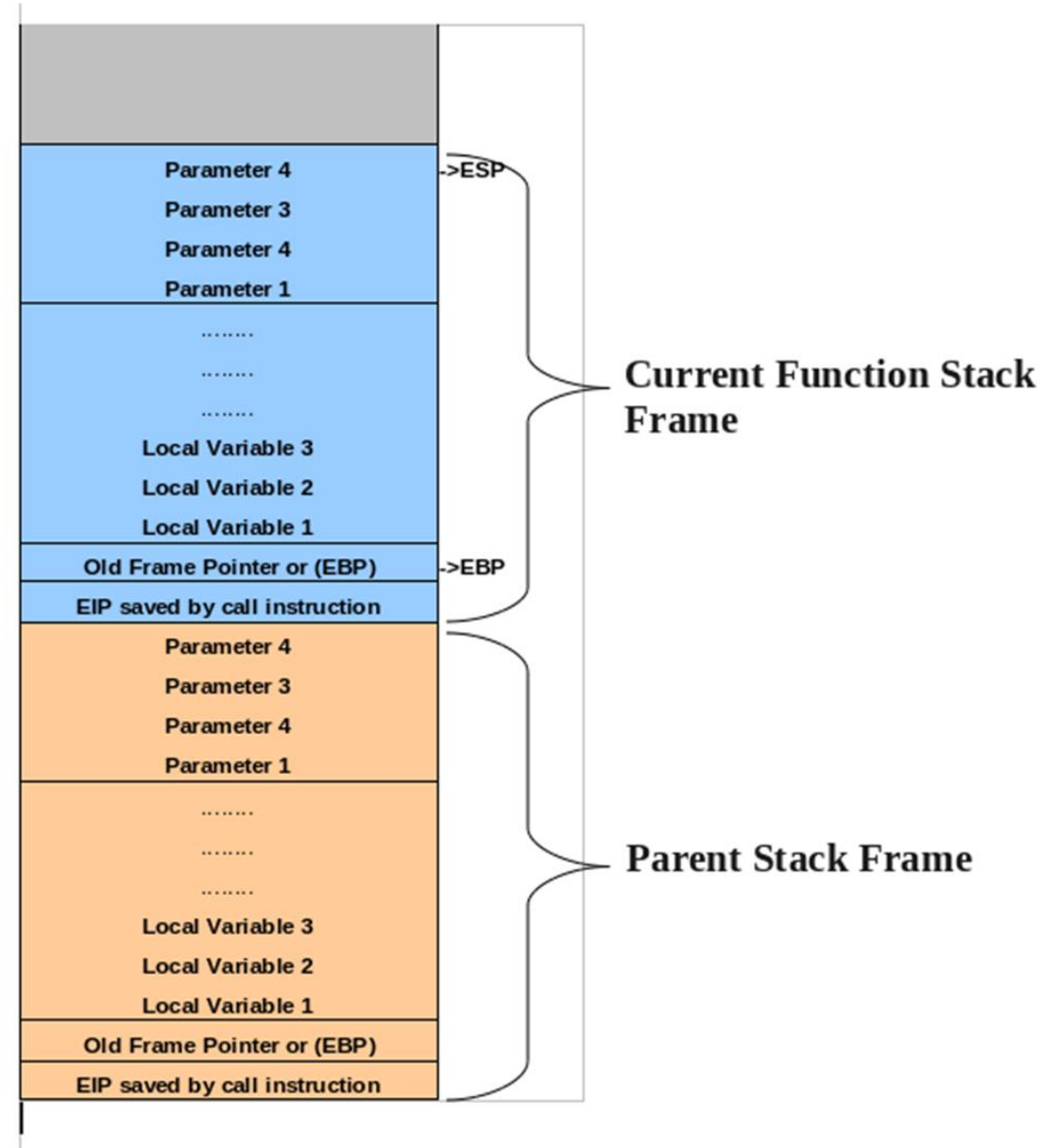
Buffer overflow – En la práctica

Variables Automáticas (locales) -> PILA Programa



Un poco de teoría: Pila de programa

- **REGISTRO EBP = BASE PILA**
- **REGISTRO ESP = TOPE PILA**
- **ALMACENA VARIABLES LOCALES**
- CADA VEZ QUE SE LLAMA A UNA FUNCIÓN:
 - Guarda **parámetros**
 - Guarda **dirección de retorno**



Buffer overflow – En la práctica

The screenshot displays a debugger interface with three main panels:

- Code Panel:** Shows the C program source code.

```
#include <string.h>

int main(int argc, char *a
{
    char buffer[10];

    printf("Introduce una ca
    scanf("%s", &buffer);

    printf("Cadena Introduci
```
- Watches (new) Panel:** A table monitoring variables.

Function arguments		
Locals		
buffer	"ôw°\201\366w\000â\375\177"	
&buffer	(char (*)[10]) 0x22ff46	char (
- Memory Panel:** Shows a memory dump starting at address 0x22ff30.

Address	Bytes
0x22ff30:	24 30 40 00 37 31 c0 77 b0 18 40 00 0e 19 40 00
0x22ff40:	b0 18 40 00 b2 17 f4 77 b0 81 f6 77 00 f0 fd 7f
0x22ff50:	3d 00 00 00 02 00 00 00 f0 ff 22 00 fd 10 40 00
0x22ff60:	01 00 00 00 f0 24 3d 00 28 2a 3d 00 00 50 40 00

Annotations: A green box highlights the bytes at 0x22ff40. A yellow box highlights the bytes at 0x22ff50. A red box highlights the bytes at 0x22ff60. Arrows point from these boxes to labels: "EBP anterior" (yellow) and "Dirección Retorno" (red).
- CPU Registers Panel:** A table showing the state of CPU registers.

Re...	Hex	Integer
eax	0x16	22
ecx	0x77c0e6ad	2009130669
edx	0x77c2cb28	2009254696
ebx	0x7ffdf000	2147348480
esp	0x22ff30	2293552
ebp	0x22ff58	2293592
esi	0x77f681b0	2012643760
edi	0x77f417b2	2012485554
eip	0x40134e	4199246

Buffer overflow – En la práctica

Preguntas ~~sin resolver~~ resueltas:

- ¿Si nuestro buffer tenía un tamaño de 10 bytes, dónde se ha escrito el resto de la información que hemos introducido?
- ¿Por qué con 25 bytes nuestro programa ha producido un error y con 20 bytes no?

Habíamos sobrescrito la dirección de retorno de la función en la pila de programa!!

The background is dark with a pattern of wavy, horizontal lines in a slightly lighter shade. A faint, dark silhouette of a person is visible on the right side, appearing to be in a dynamic pose, possibly jumping or running.

PREGUNTAS??

DEMO TIME



CONTINUARÁ...

**TO BE
CONTINUED...** 



GRACIAS!

ASK ME
ANYTHING

Antonio Morales

Twitter: @nosoyndiemas

GitHub: @antonio-morales