Trabajo Final – Descripción de resolución de Desafíos CTF

Alumno: Lucas Rivas

Materia: introducción a la Ciberseguridad

Fecha de entrega: Octubre 2025

Introducción

Este documento contiene la resolución de dos CTFs (ImaginaryCTF y Secso CTF), cada uno con tres desafios. Cada desafio inicia con un detalle que resume: número, nombre, descripción, categoría y tipo de CTF. Luego se detalla la resolución técnica.

CTF 1 – ImaginaryCTF

Reto 1

Nombre: babybof

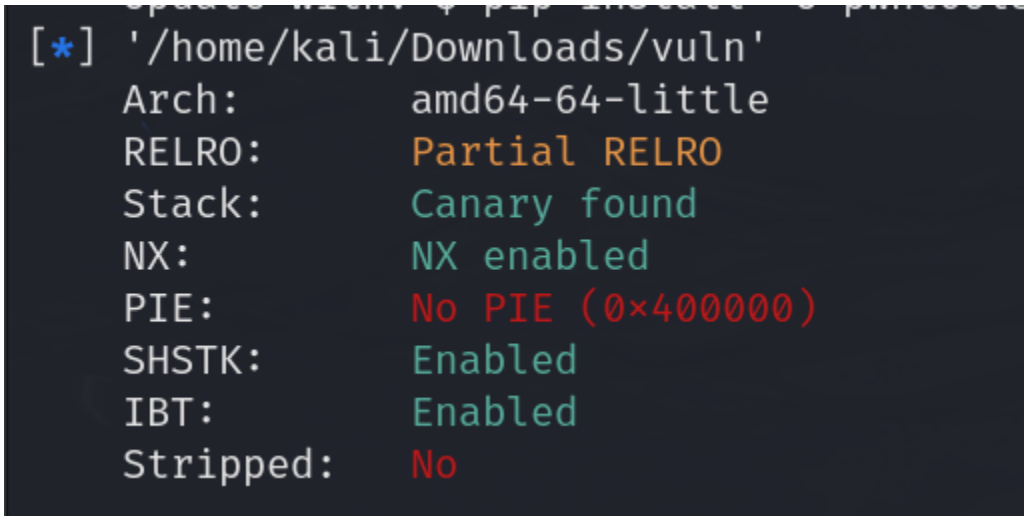
Descripción: Bienvenido a pwn. Espero que puedas solucionar tu primer desbordamiento de buffer. nc babybof.chal.imaginarictf.org 1337.

Categoría: pwn

Tipo de CTF: ImaginaryCTF

Detalle de la resolución

Utilice GDB y Pwntools para la resolución. Nos proporcionaron el binario 'vuln'. Observamos protecciones con checksec: Canary presente, NX presente, PIE deshabilitado.



El binario imprime varias direcciones útiles (system, pop rdi; ret, ret, '/bin/sh' y el canario). Con cyclic determinamos que el buffer hasta el canario era de 56 bytes. Construimos un ROP chain que preserva el canario y llama a system('/bin/sh').

Payload (resumen): A*56 + p64(canary) + B*8 + p64(ret) + p64(pop_rdi) + p64(binsh) + p64(system)

Reflexion:

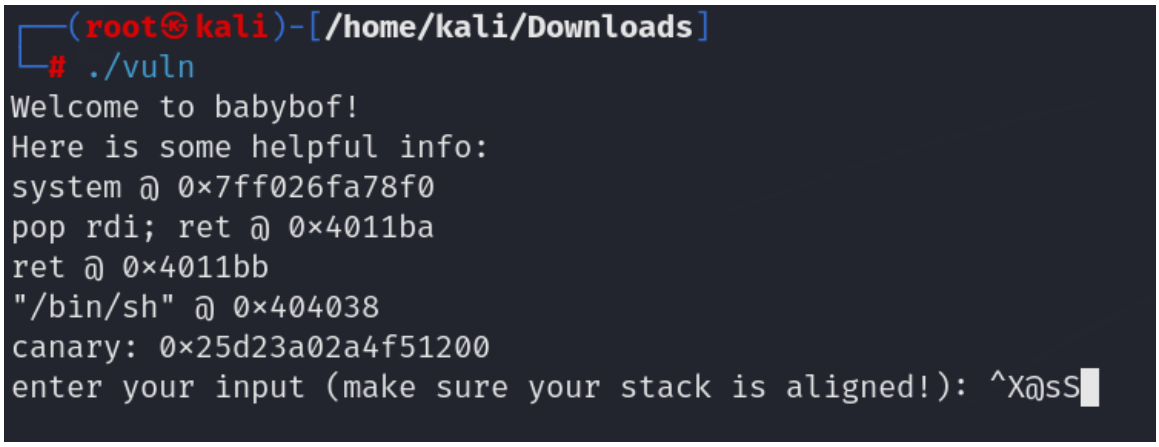
- El canario existe pero se filtra (el binario lo imprime), por lo que podemos preservarlo en la explotación.

- NX evita ejecutar shellcode en stack, por eso reutilizamos código (ROP).

- PIE está desactivado, y el binario además imprime las direcciones necesaria, por lo tanto, ya no necesitamos leakear libc ni calcular offsets dinámicos.

Añadimos ret por la **alineación** del stack para evitar problemas con ciertas instrucciones SSE en system o funciones internas.

Evidencias:



Reto 2

Nombre: comparing

Descripción: I put my flag into the program, but now I lost the flag. Here is the program, and the output. Could you use it to find the flag?

Categoría: reversing

Tipo de CTF: ImaginaryCTF

Detalle de la resolución

El programa divide la flag en pares y los introduce en una priority_queue ordenada por la suma de los ASCII de cada par. Las funciones even() y odd() generan líneas con formatos diferentes (mirror o no), y el output se produce en bloques de dos líneas por cada iteración que extrae dos pares de la cola.

La estrategia consistió en detectar si cada línea era even u odd, extraer el índice (1 o 2 dígitos), recuperar los valores ASCII y colocar cada carácter en su posición correspondiente en la flag.

Reflexion Final:

- Priority_queue como min-heap por suma: clave para entender que los pares se extraen en orden creciente de ASCII(a)+ASCII(b). No lo usamos para reconstruir el orden, pero explica por qué la salida viene en bloques coherentes de dos líneas.

- Índice i puede ser de 2 dígitos (0–15): fue la traba inicial; al permitir 1–2 dígitos el parseo se vuelve robusto.

- Validación ASCII (32–126): asegura que los splits de A (val1|val3) representen caracteres imprimibles de la flag.

- Cuando el output es una combinación de campos (A, i, espejo), conviene invertir exactamente la construcción y no adivinar con heurísticas incompletas.

- Si el índice puede crecer, tener en cuenta el número de dígitos.

- Analizar el orden de producción (dos pops-- dos líneas) evita asignaciones erróneas.

Aprendido:

- Modelar e invertir transformaciones: identificar cómo even() y odd() construyen la salida (A + i + reverse(A) vs A + i) y escribir la función inversa.

- Tratar índices y ambigüedades: el índice i no siempre es 1 dígito (0–15, 1 o 2 dígitos). Lección: no asumir; probar ambos.

- Parseo robusto con restricciones: A = str(val1)+str(val3) es ambiguo (95|48 vs 9|548). Se resuelve imponiendo rangos ASCII imprimibles (32–126).

- Comprender estructuras de datos y su efecto en el orden: la priority_queue con comparador por suma ASCII saca pares en orden particular; además, cada iteración produce dos líneas que mezclan dos pares (i1 e i2). Lección: reconstruir por bloques de 2 líneas.

- Verificación incremental: validar a mano un caso (p.ej., la primera línea) para chequear que la inversión es correcta antes de generalizar.

Evidencia lógica:

```
priority_queue<tuple<char,char,int>,...,Compare> pq;
for (int i = 0; i < flag.size() / 2; i++) {
  tuple<char,char,int> x = { flag[i * 2], flag[i * 2 + 1], i };
  pq.push(x);
}
Compare: return int(get<0>(a)) + int(get<1>(a)) > int(get<0>(b)) + int(get<1>(b));
even: A+i+reverse(A)
odd: A+i
Parsed flag: ictf{cu3st0m_c0mp@r@t0rs_1e8f9e}
```

Reto 3

Nombre: significant

Descripción: The signpost knows where it is at all times... Please find the coordinates (lat, long) of this signpost to the nearest 3 decimals.

Categoría: osint

Tipo de CTF: ImaginaryCTF

Detalle de la resolución

Se recibió una imagen de un poste con flechas que indican ciudades y distancias. Se analizaron metadatos (exiftool), se buscaron rastros de esteganografía (steghide, binwalk) y finalmente se realizó OSINT visual: búsqueda por texto visible y comparación con imágenes públicas.

Para la resolucion intentamos con diferentes tecnicas, para la obtencion de metadatos que nos ayuden a identificar las coordenadas:

- exiftool -a -gps:all -n significant.jpg
- steghide info/extract, stegseek, binwalk -- sin contenido útil para coordinadas..
- steghide info significant.jpg
- stegseek significant.jpg /usr/share/wordlists/rockyou.txt
- binwalk -e --run-as=root significant.jpg

Resultado: coordenadas reales 37.784632, -122.407939 → redondeo a 3 decimales: 37.785,-122.408

Reflexion final:

- En retos OSINT, el enunciado puede ser un factor clave: acá la frase de navegación inercial indicaba “el cartel sabe dónde está → hay que identificar el landmark real”.

- No todo JPG con “signpost” implica EXIF o estego; validar primero la hipótesis OSINT/geolocalización.

CTF 2 – Secso CTF

Reto 1

Nombre: ezwins

Descripción: how old can it be to win ? nc challenge.secso.cc 8001.

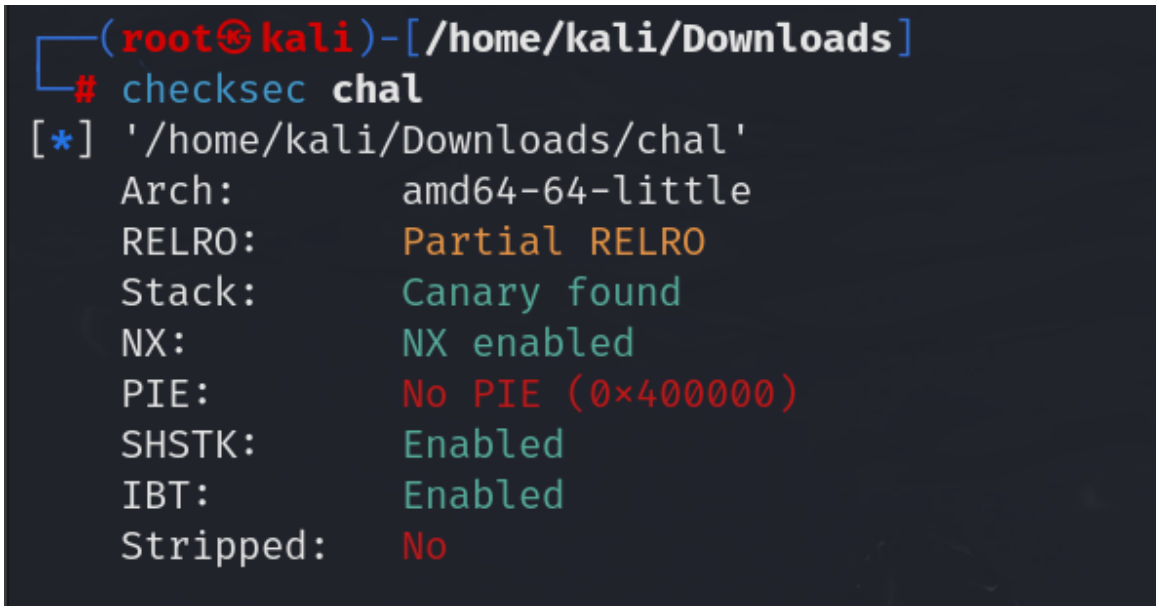
Categoría: pwn

Tipo de CTF: Secso CTF

Detalle de la resolución

El binario 'chal' mostraba un partial pointer overwrite: scanf('%d') escribía 4 bytes que se solapaban con un puntero a función guardado en la pila. Al sobrescribir los 3 bytes bajos del puntero logramos redirigir el call *%rdx a win().

Checksec:



Informacion relevante: Sin PIE ⇒ direcciones fijas. CET (IBT/SHSTK) está marcado, pero win() empieza con endbr64.
Esta información es relevante para ver como llama a sistema.

win en 0x4011f6:

00000000004011f6 <win>:

4011f6: f3 0f 1e fa endbr64

...

401208: e8 b3 fe ff ff call 4010c0 <system@plt>

→ llama a system con una cadena embebida (típico /bin/sh o comando para leer flag).

Reflexión final (lecciones aprendidas)

- Las funciones “seguras” no alcanzan si combinás mal la memoria.
  Usaron fgets con tamaño, pero mezclaron en la misma región de stack un buffer y un puntero a función desalineado, y después escribieron con scanf("%d") a +0x20 del buffer… solapando el puntero. Una sola escritura de 4 bytes bastó para desviar el control.

- Sin PIE, un *partial overwrite* basta.
  Al tener direcciones estáticas, podés “atinar” a win() cambiando solo los 3 bytes bajos. Esto evita lidiar con canario/NX y hace el exploit muy estable.

- Metodología de explotación limpia:

    - Identificar gadgets/funciones objetivo (win()).

    - Confirmar solapamiento con GDB (stack dump antes/después).

    - Diseñar el valor exacto en LE que modifica solo lo necesario.

    - Automatizar con pwntools y post-exploit (leer /flag).

Cálculo: deseamos que los bytes en el puntero sean f6 11 40 (little-endian). Preparando el entero como (0x4011f6 << 8) obtenemos 0x4011f600 = 1074790144 decimal, valor que ingresamos en scanf para lograr la sobrescritura parcial.

Evidencia (GDB):

```
gdb ./chal
(gdb) disas main
...
401258: lea    rax, [rip-0x4f]        # 0x401210 <print_greeting>
40125f: mov    [rbp-0x2f], rax        ; guarda puntero a función en la pila
401281: mov    rdx, [rip+0x2dd8]      # stdin
401294: call   fgets@plt
4012a8: lea    rax, [rbp-0x50]
4012ac: add    rax, 0x20
4012b0: mov    rsi, rax
4012bd: call  __isoc99_scanf@plt
...
40131a: call   *rdx                   ; ¡salto indirecto!
# Calculated integer to write with scanf: 1074790144 (0x4011f600)
```

Reflexión: partial overwrite + no PIE = exploit estable; CET no impidió la explotación porque win() comienza con endbr64.

Reto 2

Nombre: jumping

Descripción: I made this game in high school and I can’t seem to clear it! Can you help?

Categoría: reversing

Tipo de CTF: Secso CTF

Detalle de la resolución

El ejecutable fue empaquetado con PyInstaller. Se extrajeron los .pyc con pyinstxtractor, se identificó el código y se encontró una ofuscación por XOR entre una lista de enteros y una cadena percent-encoded. Se aplicó la operación inversa (unquote + XOR) para recuperar la flag.

Comando usado:

strings test.pyc | grep -i 'win' strings test.pyc | grep -i 'flag' strings test.pyc | grep -i 'decode'

Hallazgos:

- "You win!" → indica condición de victoria.
- "decode" → sugiere que la flag está codificada.
- Fragmentos como K17_, K170, K17F9 → posibles partes de la flag.

Aprendizaje: Los strings revelan pistas clave incluso sin decompilar el código.

Herramientas probadas:

- decompyle3
- uncompyle6

Problema: Ambas fallan por estar compilado en Python 3.10.0.

Una vez intentando y probando varias alternativas, vemos que, podemos obtener el codigo de test.pyc con: **https://tool.lu/pyc/**

Reflexion final:

- PyInstaller puede reversearse sin jugar el juego.

- Los strings y estructuras de colisión (colliderect) revelan la lógica interna.
- La flag puede estar ofuscada con XOR, pero es recuperable si se intercepta el flujo.
- Crear recursos faltantes permite ejecutar binarios que de otro modo fallan.
- CyberChef y scripts personalizados son herramientas clave para decodificación

Evidencia (extracción y

decodificación):

```
pyinstxtractor.py jumping
-> extracted test.pyc
strings test.pyc -> "You win!", "decode", fragments K17_
Decoded via XOR using key list and percent-encoded bytes:
Result: K17{F9_70_jump_no_cheat}
```

Reto 3

Nombre: dinner

Descripción: foooood!!! fooooooooooooood!!!! what is the name of this venue ?!?!?!

Categoría: osint

Tipo de CTF: Secso CTF

Detalle de la resolución

- La imagen contenía texto 'Tech Central Gala Dinner', fechas y patrocinadores. Se realizaron búsquedas con OCR y consultas dirigidas. Tras comparar visualmente con fotos públicas, se identificó el venue como Overseas Passenger Terminal – Cargo Hall (Circular Quay, Sydney).

  Busquedas realizadas utilizando Google dorks:

  - En navegadores:

    - "Tech Central" Sydney "Gala Dinner"

    - "Tech Central" "Gala Dinner" "15-17 September"

    - site:linkedin.com "Tech Central" "Gala Dinner"

    - site:instagram.com Tech Central Sydney Gala

    - site:eventbrite.com "Tech Central" Gala Sydney

Respuesta: Overseas Passenger Terminal – Cargo Hall (Sydney)

Limitaciones:

- No se contó con metadata EXIF ni fuente oficial pública del evento en el momento del análisis,utilizando técnicas como las mencionadas en el reto3 del CTF anterior.

Reflexión: El punto importante, fue la observación de la imagen y la búsqueda de información sobre la misma. Y la combinación de búsqueda para hallar el destino.