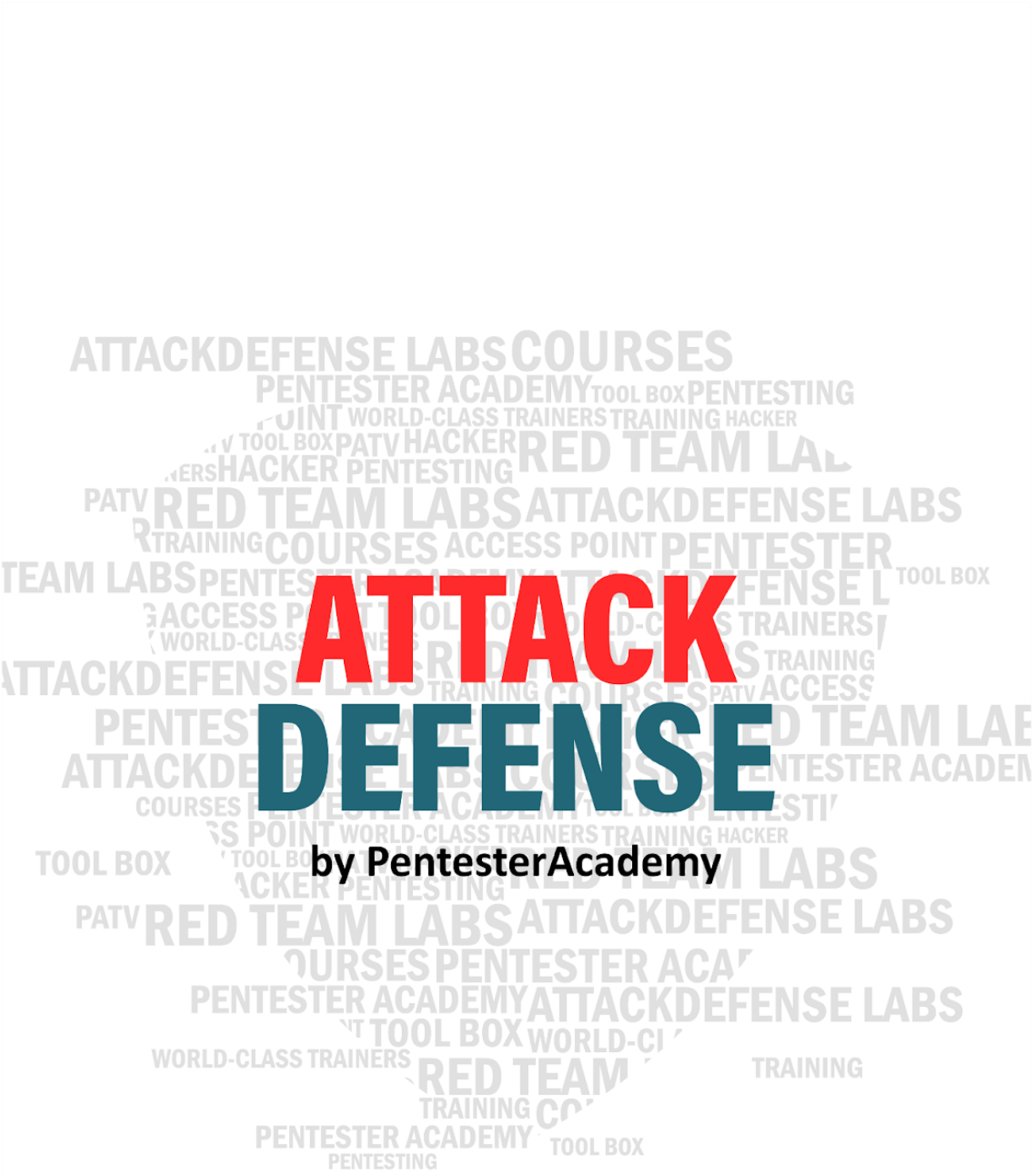

ATTACK
DEFENSE
by PentesterAcademy

Name	Malicious Binary
URL	https://attackdefense.com/challengedetails?cid=1455
Type	Docker Security : Docker Forensics

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Objective: **Study the changes made by the attacker and find the backdoor?**

Solution:

Step 1: Check running containers and run.sh is the entrypoint or init process.

Command: docker ps

```
root@localhost:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
7c595918bede        lamp-wordpress      "./run.sh"          8 minutes ago       Up 8 minutes        80/tcp              wordpress
root@localhost:~#
```

A wordpress container is running on the host.

Step 2: Check the changes made to the container after starting. Docker diff command can take container_name or container_id

Command: docker diff wordpress

```
root@localhost:~# docker diff wordpress
C /run
A /run/supervisor.sock
C /run/supervisord.pid
C /run/apache2
C /run/apache2/apache2.pid
C /run/mysqld
C /run/mysqld/mysqld.pid
A /run/mysqld/mysqld.sock
C /run.sh
C /tmp
A /tmp/do
A /tmp/sample
```

Step 3: To check Execute bash in the container in interactive mode.

Commands: docker exec -it wordpress bash

```
root@localhost:~# docker exec -it wordpress bash
root@7c595918bede:/#
```

Step 4: Check the contents of the run.sh file

Command: cat run.sh

```
root@7c595918bede:/# cat run.sh
#!/bin/bash

VOLUME_HOME="/var/lib/mysql"

sed -ri -e "s/^upload_max_filesize.*/upload_max_filesize = ${PHP_UPLOAD_MAX_FILESIZE}/" \
    -e "s/^post_max_size.*/post_max_size = ${PHP_POST_MAX_SIZE}/" /etc/php5/apache2/php.ini
if [[ ! -d $VOLUME_HOME/mysql ]]; then
    echo "=> An empty or uninitialized MySQL volume is detected in $VOLUME_HOME"
    echo "=> Installing MySQL ..."
    mysql_install_db > /dev/null 2>&1
    echo "=> Done!"
    /create_mysql_admin_user.sh
```

```
else
    echo "=> Using an existing volume of MySQL"
fi

exec supervisord -n
/tmp/sample
root@7c595918bede:/#
```

The last line seems to be calling sample binary kept in /tmp directory.

Step 5: Check the file information of “sample” binary.

Command: file /tmp/sample

```
root@7c595918bede:/# file /tmp/sample
/tmp/sample: ELF 64-bit LSB  shared object, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 3.2.0, BuildID[sha1]=
7423b1a72d6109ee43a2a4813fdb5f2fcf98597a, not stripped
root@7c595918bede:/#
```

Step 6: Copy the “sample” binary to host machine.

Command: docker cp wordpress:/tmp/sample .

```
root@localhost:~# docker cp wordpress:/tmp/sample .
root@localhost:~# ls -l
total 16
-rwxr-xr-x 1 root root 15040 Dec  1 10:17 sample
root@localhost:~#
```

Step 7: Analyze the “sample” binary with gdb and set the disassembly syntax as intel

Commands:
gdb -q ./sample
set disassembly-flavor intel

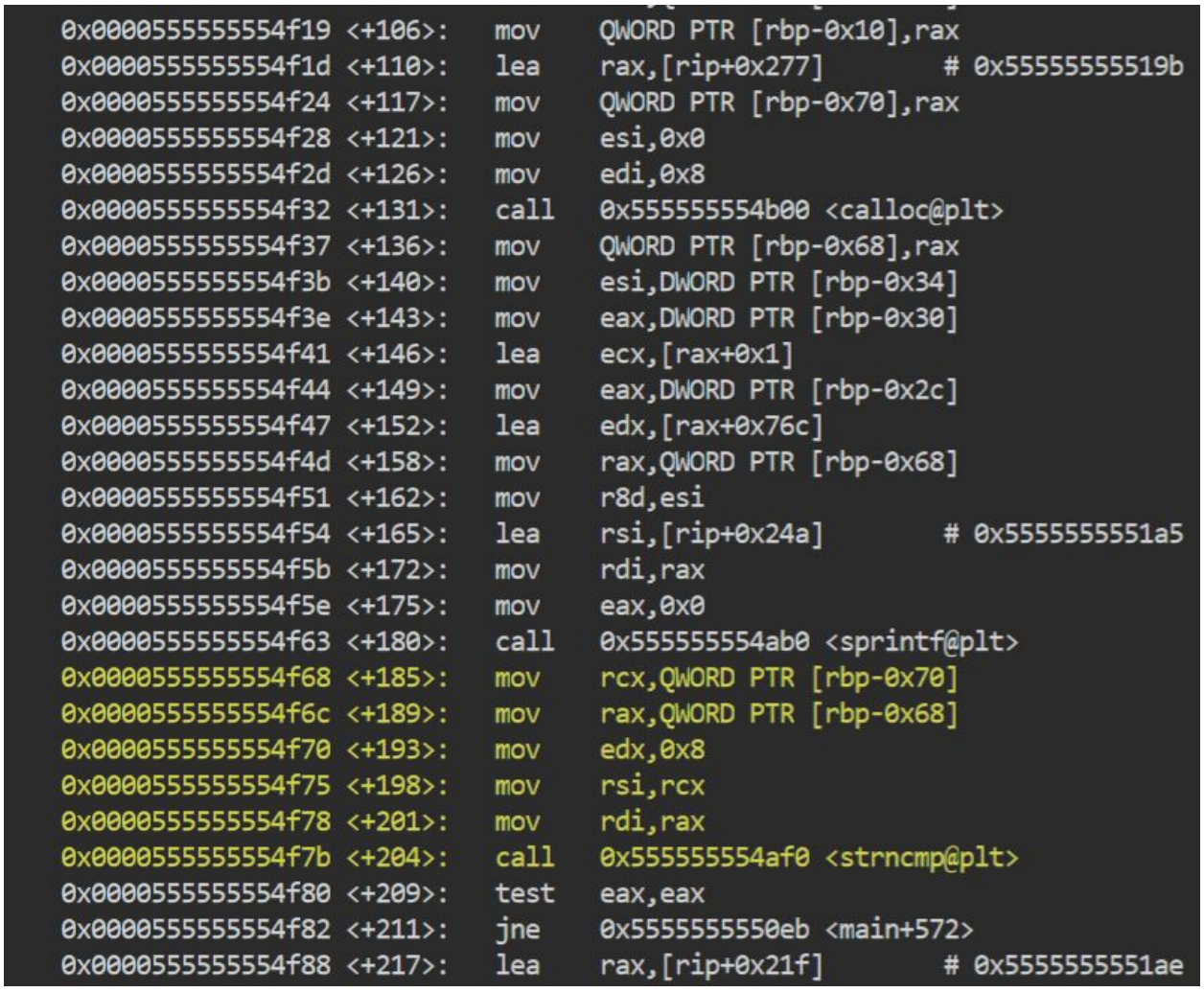
```
root@localhost:~# gdb -q ./sample
Reading symbols from ./sample...(no debugging symbols found)...done.
(gdb)
(gdb)
(gdb) set disassembly-flavor intel
```

Step 8: Disassemble the main function. Use start to place a temporary breakpoint at the beginning of the main function.

Commands:
start
disas main

```
(gdb) start
Temporary breakpoint 1 at 0xeb3
Starting program: /root/sample
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Temporary breakpoint 1, 0x0000555555554eb3 in main ()
(gdb) disas main
Dump of assembler code for function main:
   0x0000555555554eaf <+0>:     push   rbp
   0x0000555555554eb0 <+1>:     mov    rbp,rsp
=> 0x0000555555554eb3 <+4>:     sub    rsp,0x90
   0x0000555555554eba <+11>:    mov    rax,QWORD PTR fs:0x28
   0x0000555555554ec3 <+20>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000555555554ec7 <+24>:    xor    eax,eax
   0x0000555555554ec9 <+26>:    mov    edi,0x0
   0x0000555555554ece <+31>:    call   0x555555554b20 <time@plt>
   0x0000555555554ed3 <+36>:    mov    QWORD PTR [rbp-0x78],rax
   0x0000555555554ed7 <+40>:    lea    rax,[rbp-0x78]
   0x0000555555554edb <+44>:    mov    rdi,rax
   0x0000555555554ede <+47>:    call   0x555555554b10 <localtime@plt>
```



The strings which are to be compared are stored in rsi and rdi registers. The string compare operation occurs on line 204.

Step 9: After analyzing the assembly code, set a breakpoint at the call to strncmp function.

Command: break *main +204

Continue the execution until the breakpoint is hit.

Command: continue

```
(gdb) break *main +204
Breakpoint 2 at 0x555555554f7b
(gdb) continue
Continuing.

Breakpoint 2, 0x0000555555554f7b in main ()
```

Step 10: Examine the arguments of the function.

Commands:
x/s $rdi
x/s $rsi

```
(gdb) x/s $rdi
0x555555770860: "2019-12-1"
(gdb) x/s $rsi
0x55555555519b: "2012-2-29"
```

The analysis of the main function reveals that it checks if date is “2012-2-29”.

Now, we have two ways to approach it:

Approach I: Set system date

Step 11: Set the system date to “2012-2-29”.

Command: date +%y-%m-%d -s "20120229"

```
root@localhost:~# date +%y-%m-%d -s "20120229"
12-02-29
root@localhost:~#
```

Step 12: Run the sample binary

Command: ./sample

```
root@localhost:~# ./sample

* Could not resolve host: ftp.imaginary-67823-url.com
* Closing connection 0
Curl error 6
root@localhost:~#
```

The binary tries to connect to ftp.imaginary-67823-url.com domain.

URL: ftp.imaginary-67823-url.com

Approach II: Continue in GDB

Step 11: Get past the date check condition by setting the instruction pointer to the address of the instruction after the date check (<main + 217>). Modify the instruction pointer to contain the address of instruction at <main + 217>.

Commands:
set $rip=*main+217
continue

```
(gdb) set $rip=*main+217
(gdb) continue
Continuing.
[New Thread 0x7ffff199c700 (LWP 1798)]
[Thread 0x7ffff199c700 (LWP 1798) exited]
* Could not resolve host: ftp.imaginary-67823-url.com
* Closing connection 0
Curl error 6
[Inferior 1 (process 1741) exited normally]
(gdb)
```

The binary tries to connect to ftp.imaginary-67823-url.com domain.

URL: ftp.imaginary-67823-url.com