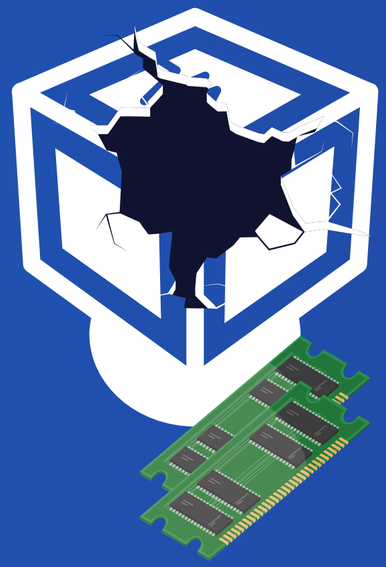
HACK IN BO®
Spring 2023 Edition
20ª EDIZIONE



TAMPERING THE RAM

A NEW WAY TO BREAK INTO ANY VM

Presented by UNLOCK SECURITY

@mrnfrancesco

FRANCESCO MARANO

ETHICAL HACKER | PENETRATION TESTER

eWPT eCPPT eMAPT

"I love making software do things other than what they were designed to do"

Penetration Test IT
Web, Mobile, Network

Penetration Test OT
Automotive, Naval Systems, Industrial Machineries

Android Kernel Exploitation & Post-Exploitation

HACK IN BO®
Spring 2023 Edition
20ª EDIZIONE

Think secure

UNLOCK SECURITY

www.unlock-security.it

Sample use case

We want to get access to a VM being run locally with:

01 **Encrypted disk**
No possibility to mount the disk externally

02 **No editable boot parameters or recovery mode**
Grub configuration doesn't allow to start a root shell

03 **Uknown users**
No username is known and the password policy is very strong

04 **No service exposed**
No web server or SSH, the VM is clean and only exposes login

Page cache theory

Page Cache is part of the Virtual File System (VFS) that is meant to improve I/O performance in read/write operations.

This is referred to as page caching because the kernel works with memory units called pages.

So instead of tracking and managing each individual bit of information that the system accesses, the entire page containing that information is managed.

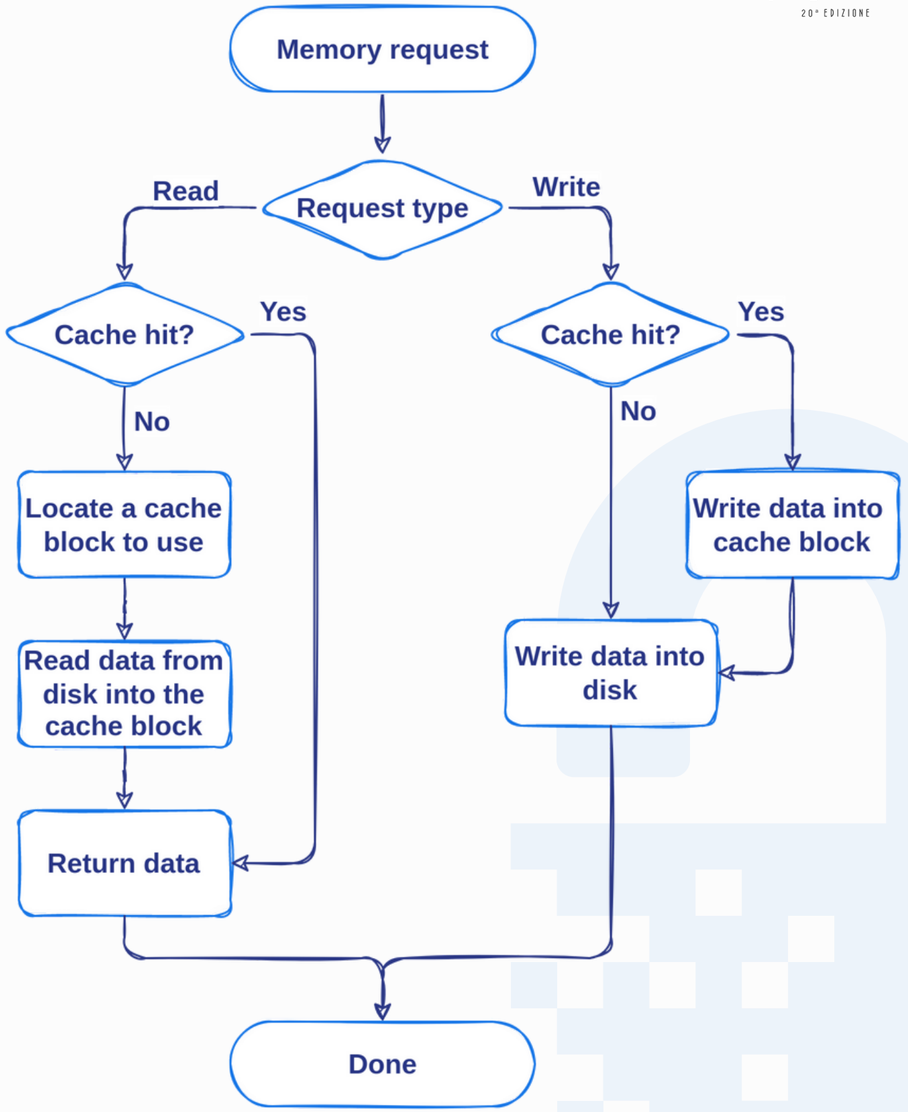
Pages are generally 4 KB in size, and since they represent the fundamental unit of Page Cache, it means that every request to read even a single bit will result in 4 KB of data being read.

Read request workflow

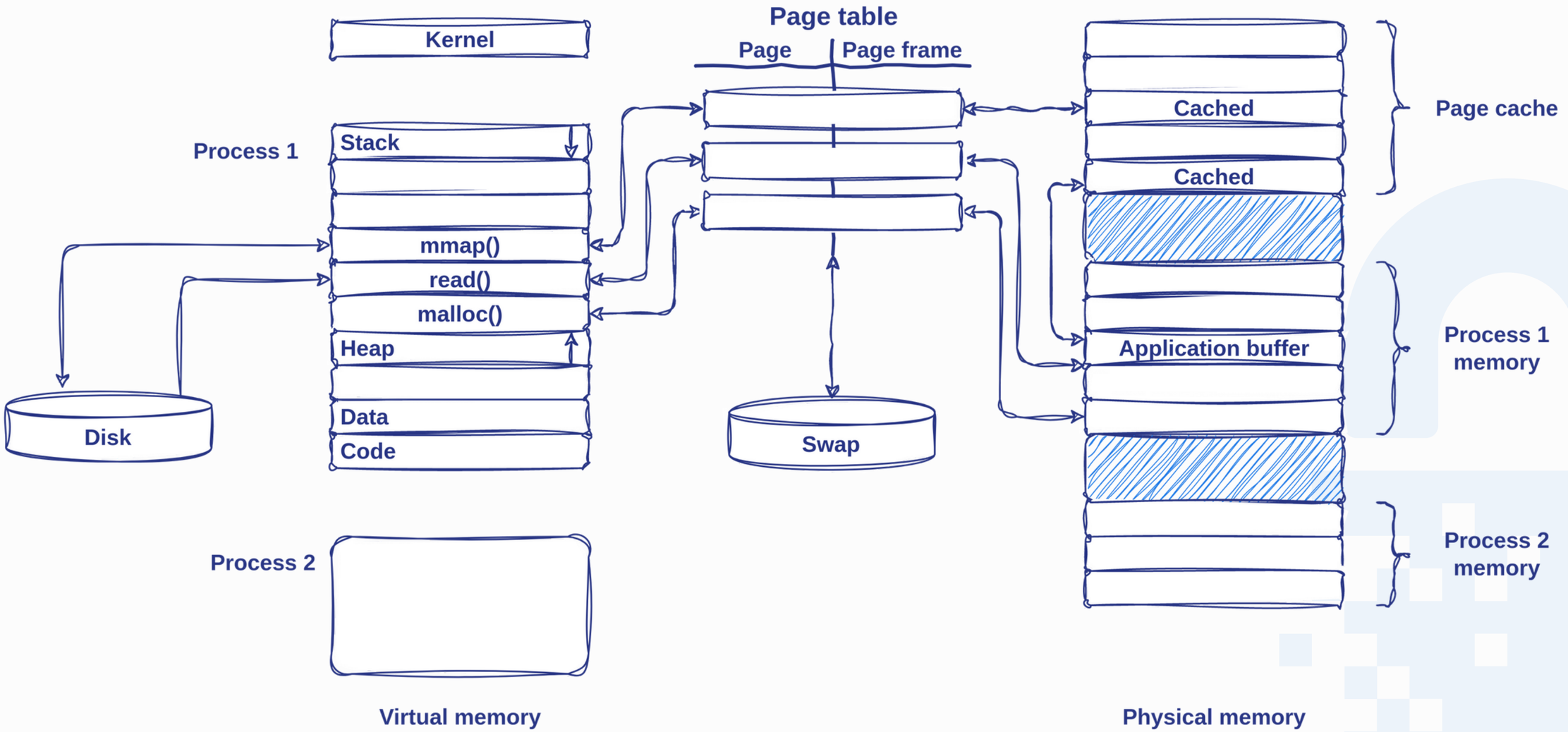
1. A user-space application requests the kernel to read data from disk using a system call like read(), mmap(), etc.

2. The kernel checks whether the pages corresponding to the requested data are already in the cache and if so (Cache HIT) returns them directly to the caller.
In this case the disk is never accessed.

3. If not (Cache MISS), the kernel reserves enough space in the cache to store them, then performs a read of the data from disk. The pages are then cached and returned to the caller.
From now on, each request to access the file will generate a Cache HIT and won't involve the disk.



Read request workflow



Page cache practice

By creating a 128 MB test file we are going to test how the cache works during read operations. We clear all caches in order to start clean. We can make sure the test file has been cleared from the cache by using the vmtouch by checking the "Resident Pages" count is zero.

```
$ dd if=/dev/random of=file count=128 bs=1M

$ sync

$ echo 3 | sudo tee /proc/sys/vm/drop_caches

$ vmtouch file
           Files: 1
     Directories: 0
  Resident Pages: 0/32768  0/128M  0%
         Elapsed: 0.002936 seconds
```

Reading file with read() syscall

We create a simple C program that reads 2 bytes of the test file.

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>

int main(int argc, char *argv[])
{
 int fd = open("file", O_RDONLY);
 uint8_t buf[2] = {0};
 read(fd, buf, sizeof(buf));
 close(fd);

 return 0;
}
```

Reading file with read() syscall

We expect to cache the entire page (4 KB) containing the two bytes read, but we cache 12 (48 KB). This is because the kernel implements the read ahead logic, whereby data not requested, that is likely to be accessed in the near future, is loaded into memory in order to reduce disk accesses.

```
$ gcc -Wall -o test-read test-read.c

$ ./test-read

$ vmtouch file
           Files: 1
     Directories: 0
  Resident Pages: 12/32768 48K/128M 0.0366%
         Elapsed: 0.00183 seconds
```

Reading file with read() syscall

This behavior can be controlled using the posix_fadvise() syscall.

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>

int main(int argc, char *argv[])
{
 int fd = open("file", O_RDONLY);
 posix_fadvise(fd, 0, 0, POSIX_FADV_RANDOM);
 uint8_t buf[2] = {0};
 read(fd, buf, sizeof(buf));
 close(fd);

 return 0;
}
```

Reading file with read() syscall

We clear the cache, compile and run the program, then check the cache status again.

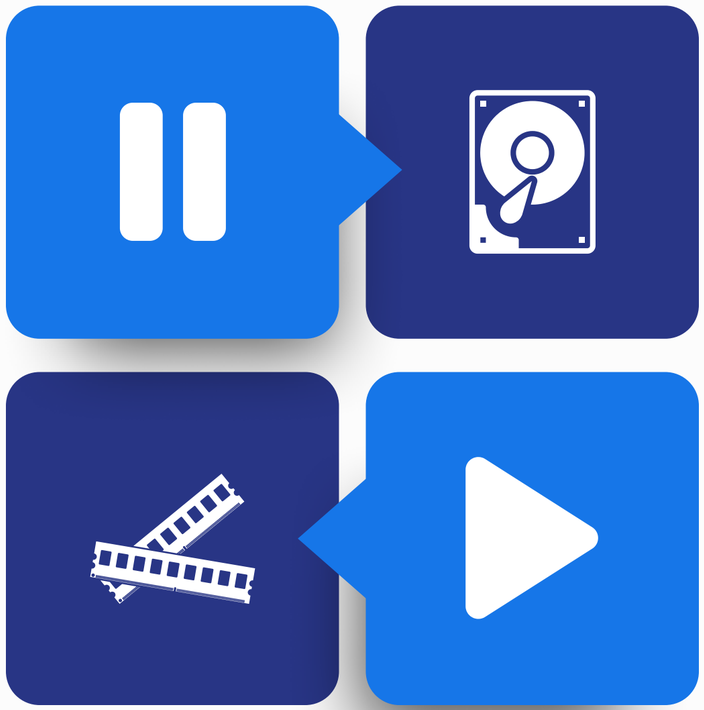
```
$ sync; echo 3 | sudo tee /proc/sys/vm/drop_caches

$ gcc -Wall -o test-fadvise test-fadvise.c && ./test-fadvise

$ vmtouch file
           Files: 1
     Directories: 0
  Resident Pages: 1/32768  4K/128M  0.00305%
         Elapsed: 0.001783 seconds
```

Virtual Machine paused state

When we use a virtualization system it is possible to pause the execution of the VM. When we do this the state of the VM is saved to disk. The pause function is useful when we want to temporarily stop the execution of an operation on the VM to resume it later.



When we use a VM some files are created in the host system:

- vmdk – virtual disk equivalent to the VM's disk
- vmsd – metadata and other info about the VM snapshots
- vmx – configuration info and settings of the VM hardware
- vmsn – VM status related to a particular snapshot
- vmem – memory contents when pausing the VM

The vmem file is particularly interesting because it allows us to analyze the contents of the RAM of a paused VM; contents that will not change as long as the VM remains paused.

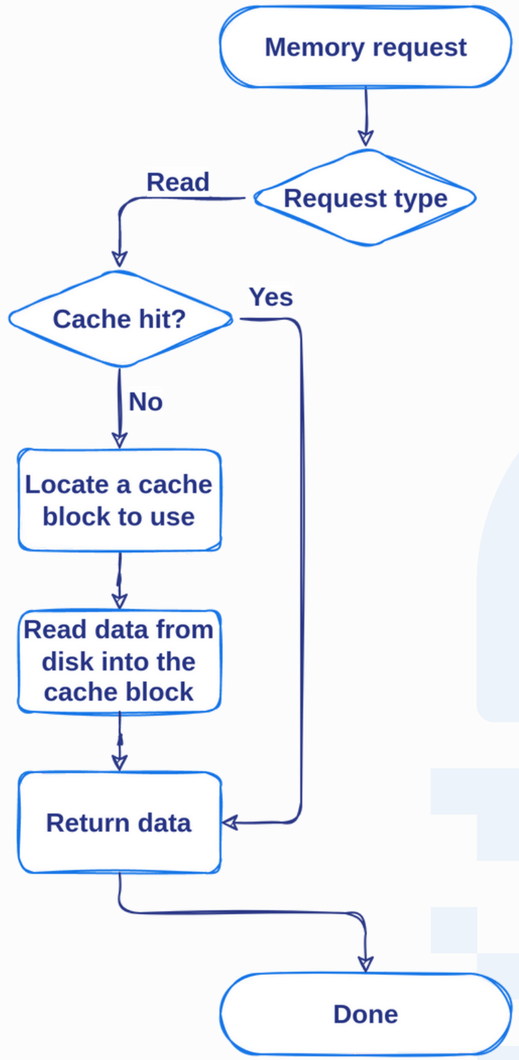
Did you get the point?

This is the workflow for the attack to work:

01 **Start the VM**
Interact with it to force a file read

02 **Pause the VM**
Edit the Page Cache from the vmem file

03 **Resume the VM**
Force again the file read bypassing the disk



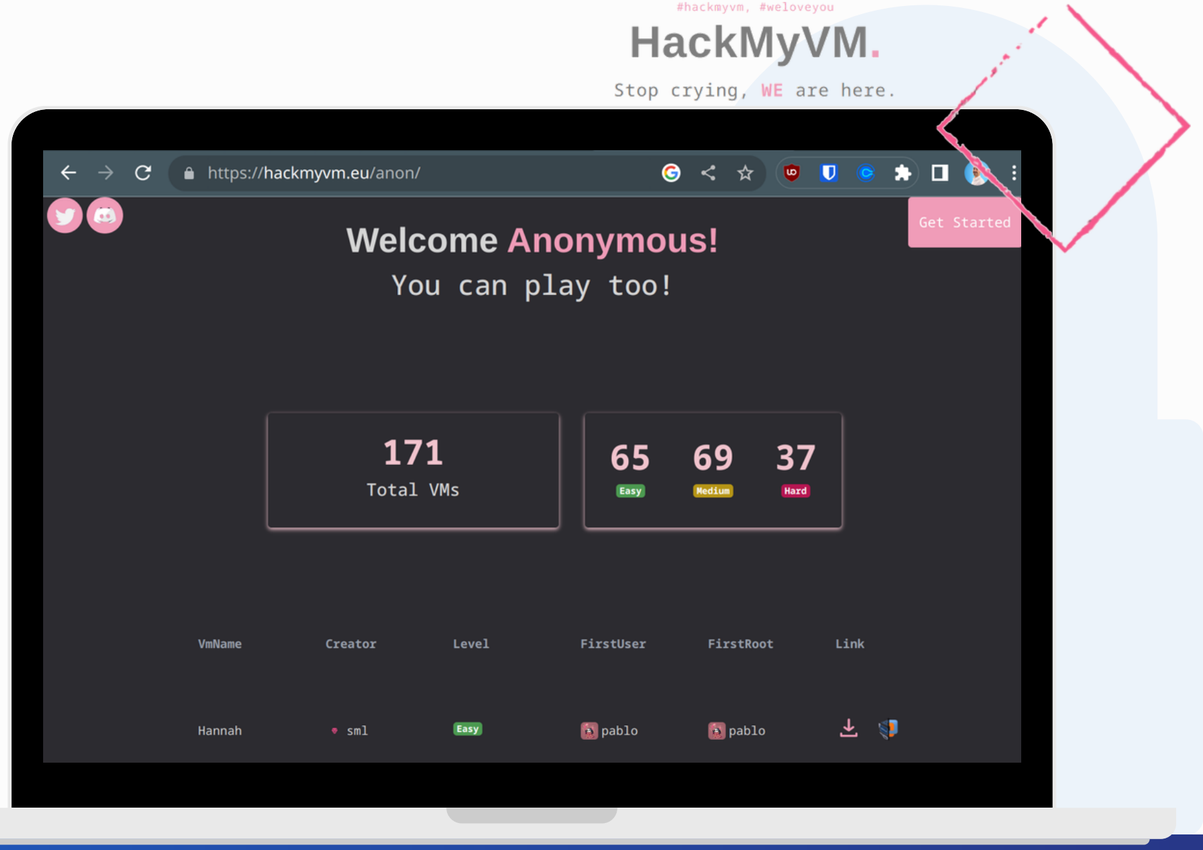
Let's play with it

We will use "Hannah" CTF from HackMyVM.

01 **Username enumeration**
Force reading of /etc/passwd file

02 **Passwords hashes enumeration**
Force reading of /etc/shadow

03 **Login bypass & privilege escalation**
Force again the file read bypassing the disk



Username enumeration

Once the VM is started all we have is the login screen. The first goal is to find at least one valid user on the system. In Linux the list of users along with their other properties is in the /etc/passwd file.

1. What we do know is that in front we have the /usr/bin/login binary. To simplify, this is a program executed indirectly by the init process via getty or a display manager.

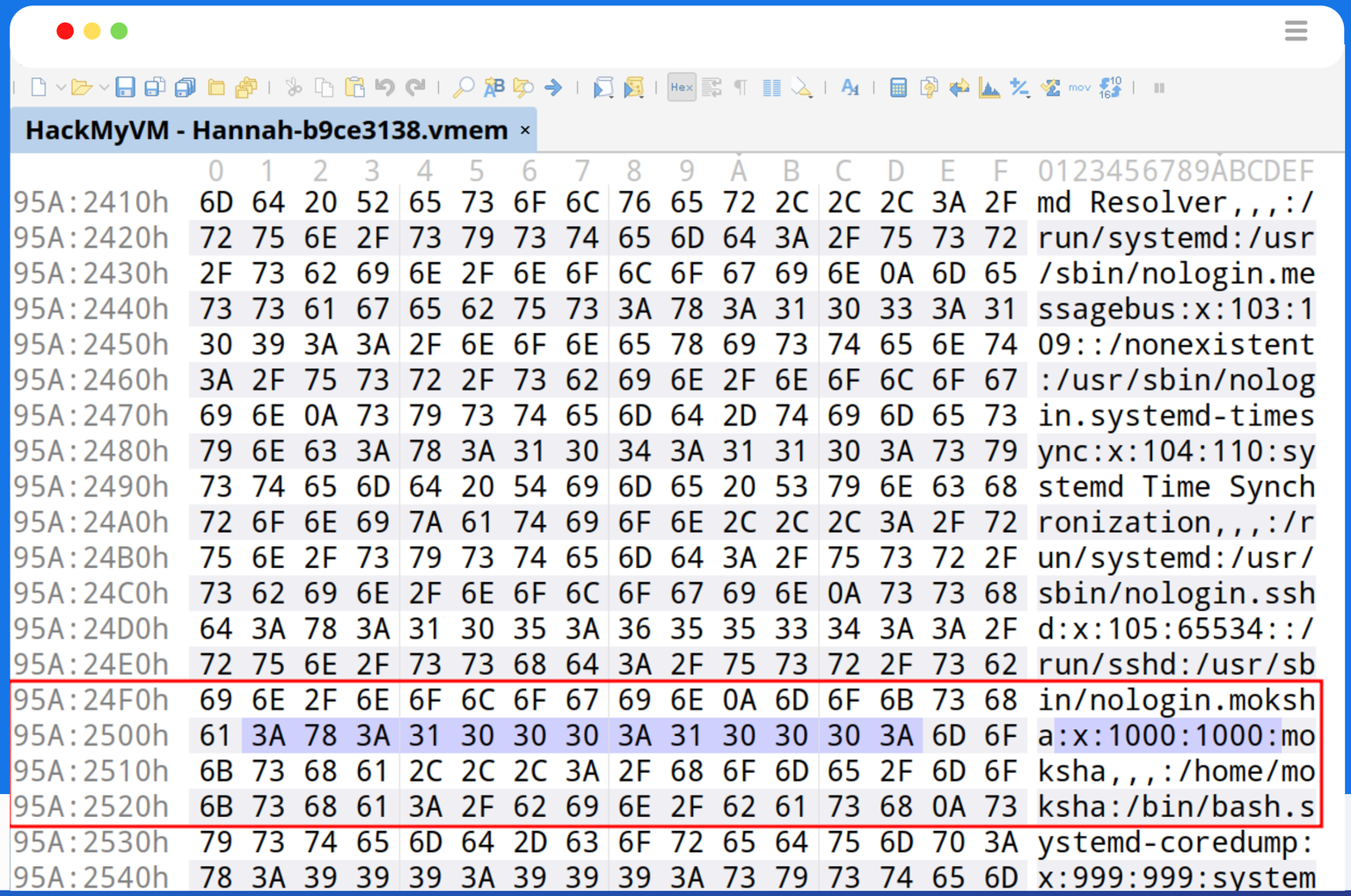
2. When executed it prompts the user for the username and password to login with. The username entered is checked for existence in the /etc/passwd file before eventually checking the password.

3. Just try to login using totally random credentials then pause the VM and open the vmem file using an hex editor.

Username enumeration

We look for matches of the string :x:1000:1000: that is, the line in the /etc/passwd file that refers to the user with ID 1000 (generally used as the first ID for users created on Linux).

We find the user moksha.

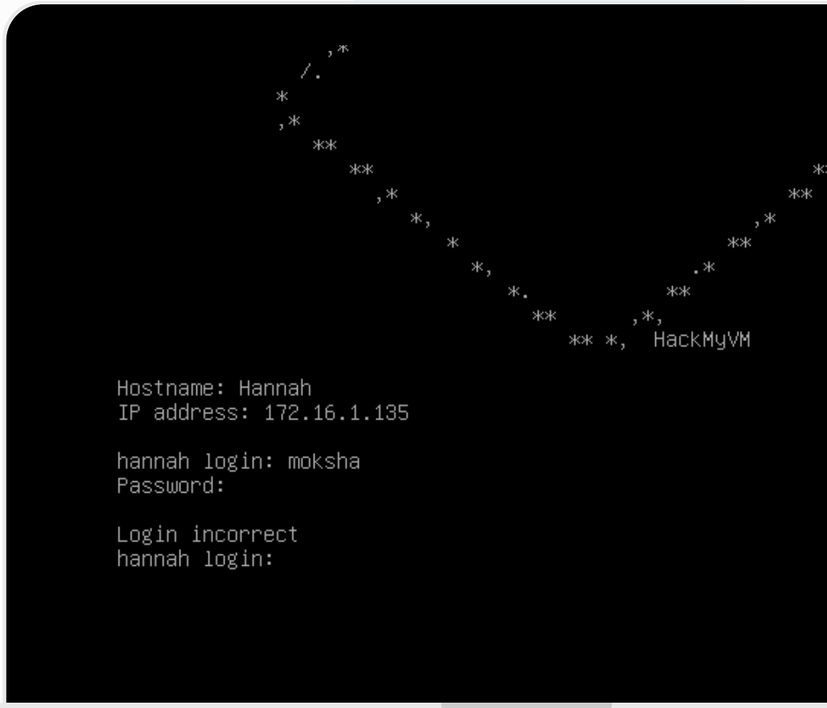


Passwords hashes enumeration

Suppose we want to log in with the user moksha, then we want to look in the vmem file for a match for the string moksha:$, that is, the beginning of the line in the /etc/shadow file corresponding to the hash of the moksha user's password.

1. If we tried immediately we would not get any results. This is because the /etc/shadow file is accessed only when a login attempt is made with a correct username.

2. Then we unpause the VM, enter the moksha username and a randomly chosen password to force read the files /etc/passwd (from cache, CACHE HIT) and /etc/shadow (from disk, CACHE MISS).

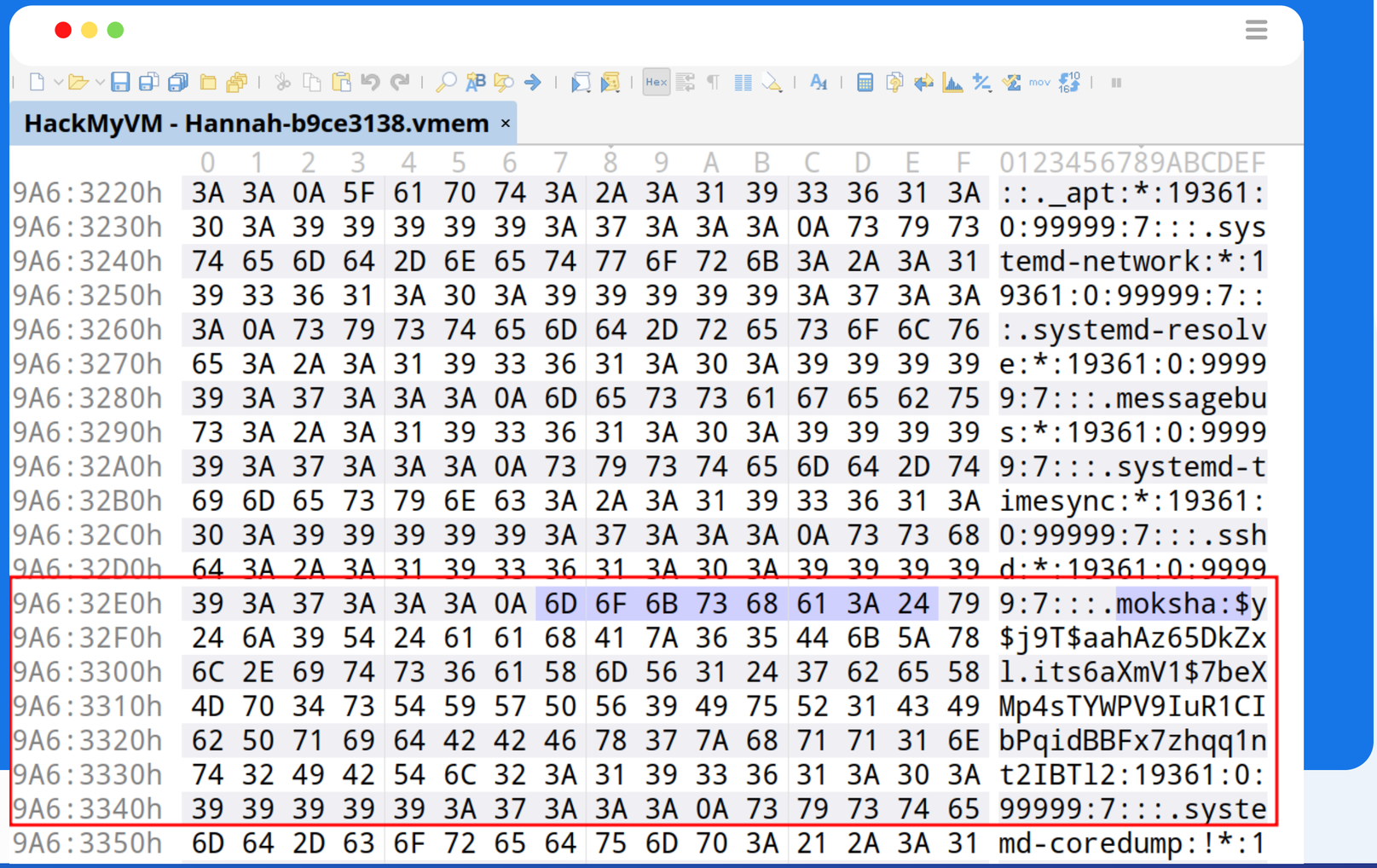


Passwords hashes enumeration

We pause the VM once again and look for matches of the string moksha:$.

We find a yescrypt hash:

moksha:yj9T$aahAz65DkZxl.its6aXmV1$7beXMp4sTYWPV9IuR1CIbPqidBBFx7zhqq1nt2IBTl2:19361:0:99999:7:::



Creating a new password

yescrypt is not yet supported by openssl, so we use python to generate a new hash for a password.

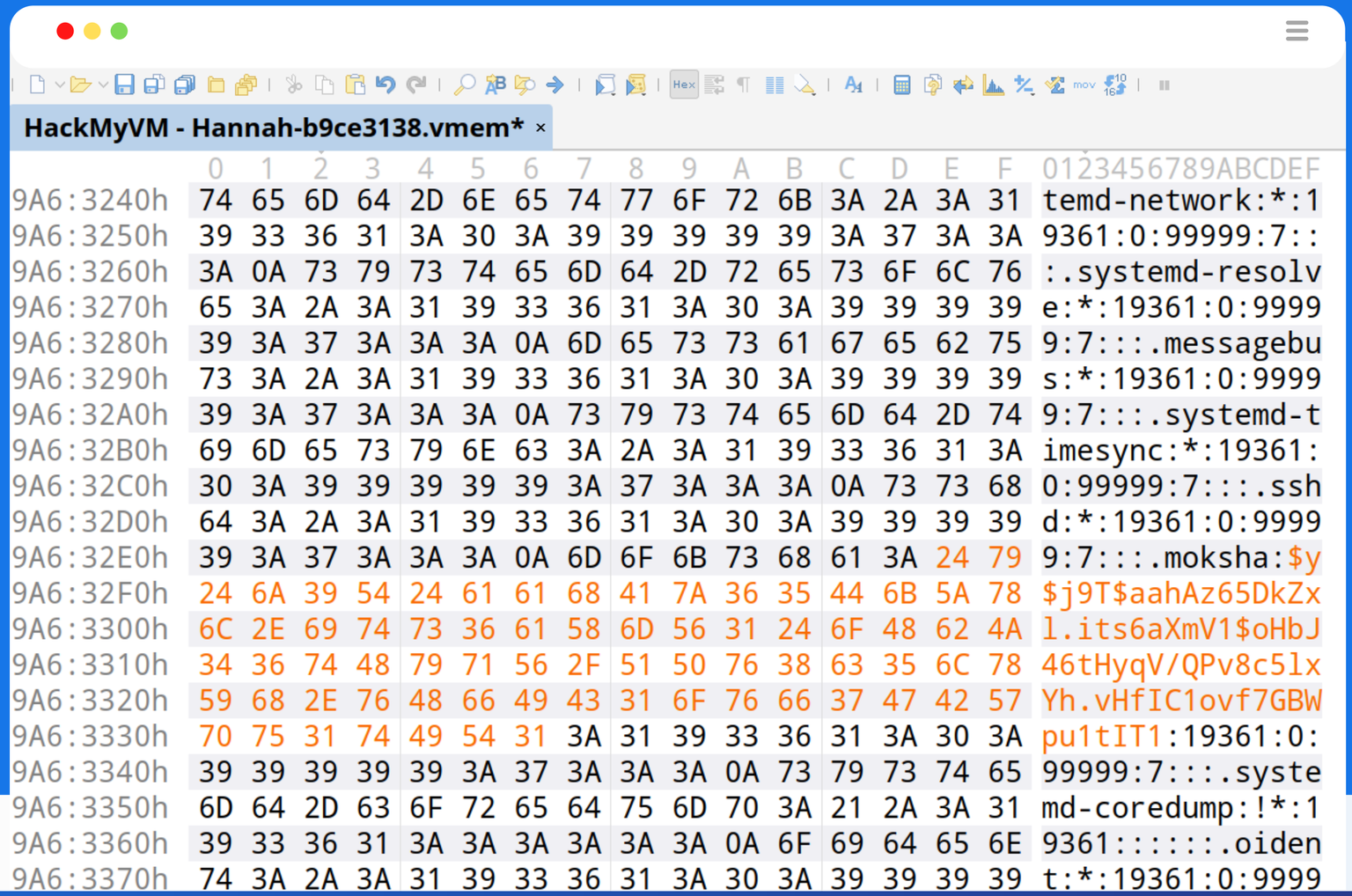
```
>>> import crypt
>>> crypt.crypt(
...     'password', # Our password of choice
...     '$y$j9T$aahAz65DkZxl.its6aXmV1$' # The salt found in the vmem file
... )
'$y$j9T$aahAz65DkZxl.its6aXmV1$oHbJ46tHyqV/QPv8c5lxYh.vHfIC1ovf7GBWpu1tIT1'
```

Login bypass & privilege escalation

We just replace the password hash with the one we generated to bypass the login.

Then we replace every :1000:1000: with :0000:0000: to do a privilege escalation.

Finally we unpause the VM and log in with credentials moksha:password.



Login bypass & privilege escalation

HACK IN BO®
Spring 2023 Edition
20ª EDIZIONE

LIVE DEMO

Presented by UNLOCK SECURITY

@mrnfrancesco

FRANCESCO MARANO

Contact me

HACK IN BO®
Spring 2023 Edition
20ª EDIZIONE

Think secure

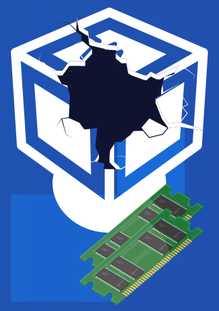
UNLOCK SECURITY

www.unlock-security.it

Think secure

UNLOCK SECURITY

Thank you!



TAMPERING THE RAM

A NEW WAY TO BREAK INTO ANY VM

HACK IN BO®

Spring 2023 Edition

20ª EDIZIONE