# VM-6112819913404339

*Colander*

**Casalino Valerio, Mijalkovic Jovana, Vendrame Mario Tobia**

## CONSIDERATIONS

The machine has 2 interfaces, 1 configured with NAT to access the internet, and the second is configured to communicate only with the host at the static IP address: **192.168.56.100**. The "main" user is **tux**, which is the only one able to use sudo for every command. We didn't intend the attacker to guess the password of this user, but we set weak credentials to get surprised ("1a2b3c4d").

### USEFUL DATA

- Hostname: colander
- IP address: 192.168.56.100
- Users: [ tux:1a2b3c4d (ssh key has password 1234567890) ] [ max:abygurl69 ] [ webadmin:!holioday ] [ dockerman:#1badboy ] [ sysadmin:asdfsuperadmin ]
- Web roots are in `/var/www/html` and SSL certs in `/etc/ssl/crt`
- MySQL: wordpressuser:supersecretpasswordlol@localhost/wordpress
- MongoDB: mongodb://admin:iammongoadmin@localhost/production

## (INTENDED) USER OWN PATHS

### (1/4) PORT 21 (FTP) → TUX

**Exploit type**: service misconfiguration & ignorant user

It is realistic because security unaware users are likely to prioritize comfort over security. In fact, service misconfigurations are one of the most common vulnerabilities found in real systems.

#### Configuration

Using the `vsftpd` package, some tweaks in the configuration are needed to make the daemon accept remote connections and to allow the anonymous user. Our `/etc/vsftpd.conf` at this link : https://pastebin.com/BSe8Txrk.

#### How to exploit

Anonymous login in FTP on standard port 21. There is a backup folder, named "*super-secret!! !*", which contains the ssh private key of the user **tux**. It is password protected (pass="1234567890"). **JohnTheRipper** cracked it in no time. Next step is to ssh into tux.

### (2/4) PORT 80 (APACHE) - NO VHOST → MAX

**Exploit type**: security through obscurity & not sanitized input in PHP

It is realistic if we consider a production site misconfigured to be accessible. The most common website attacks (SQL injection, XSS, RFI, etc.), in real systems, all have the same root in common, which is input sanitization. To be more accurate, the lack of it.

#### Configuration

We configured apache to have different web roots for different protocols and virtual hosts. In this web root, we wrote `/gallery/index.html` (link here). It calls a vulnerable php file to display the image: https://pastebin.com/Qmm1GNRB.

#### How to exploit

On port 80, if looking for the raw IP, there is an empty page, but fuzzing around is possible to find `/gallery/index.html`. In this page, an image is included using a php script that can be exploited to print every file readable by `www-data`, that includes

`/etc/passwd`, in which it is possible to look at the hash of user max's password. Again, use JohnTheRipper and ssh into the system.

## (3/4) PORT 80 (APACHE) - VIRTUAL HOST → WEBADMIN

**Exploit type**: noSQL injection & password reuse

This is realistic because people often make their own login page, and as said before, SQL and noSQL injections are very common in real systems, when the user doesn't do proper input sanitization.

### Configuration

We installed mongodb, created a database with some random credentials and a real one, then we set up a login page in php that doesn't sanitize input properly. Here's the link to the login page code: https://pastebin.com/4vcS9X7L.

### How to exploit

Looking at port 443, it is using a self signed certificate for "colander.local", so, after editing `/etc/hosts`, looking at port 80, there is a login form, which is vulnerable to noSQL (MongoDB) injections. From there it is possible to enumerate usernames and passwords with a script that injects a regular expression match (eg: `&username[$regex]=".*"`). The only user that is useful is webadmin, which has the same password for his account. Now ssh to webadmin.

## (4/4) PORT 443 (APACHE) → MAX

**Exploit type**: CVE in Wordpress plugin (unauthenticated arbitrary file upload)

This vulnerability is realistic because server admins who use wordpress frequently install plugins, and sometimes those plugins have security vulnerabilities which they are not aware of.

### Configuration

We installed wordpress, then we generated a self-signed certificate with `openssl`:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/ssl/private/apache-selfsigned.key \
-out /etc/ssl/certs/apache-selfsigned.crt
```

We put in the certificate hints for the virtualhost to use. Then we installed the Reflex

Gallery Plugin v3.1.3 for Wordpress. That is a vulnerable version that allows the upload of whatever file (even a web shell).

#### How to exploit

On port 443 there is a wordpress instance with a particular plugin that can be used to upload a webshell to read `/etc/passwd` and proceed as in point 2. exploit-db link.

## (INTENDED) PRIVESC PATHS

### (1/5) TUX EXPLOITING JOURNALCTL EXECUTED AS ROOT

This is realistic because sometimes, to be able to perform some tasks quickly, some commands are allowed to be executed with higher privileges.

#### Configuration

Editing the sudoers file: `/etc/sudoers` using `visudo`, it is sufficient to allow a particular user to run `sudo journalctl -t 4` without a password adding this line:

```
tux ALL=(root) NOPASSWD:/usr/bin/journalctl -t 4
```

#### How to exploit

The user can run `sudo journalctl -t 4` without password, so by reducing the size of the terminal, the program will trigger `less` as the superuser. In less, a shell can be spawned typing `!/bin/bash`. And then you have a root shell.

### (2/5) MAX LOGGING IN AS DOCKERMAN, WHICH IS IN THE DOCKER GROUP

This is realistic because it is so convenient to be in the docker group to manage containers.

#### Configuration

It's enough to add the user in the docker group:

```
sudo usermod -aG docker dockerman
```

Then, pull a docker image to give an hint:

```
docker pull debian:10-slim
```

### How to exploit

Max has a note from tux in his home folder, with credentials for this dockerman user. If you find out that dockerman is in the docker group, then the path is simple. Using debian:10-slim (the only image already pulled from docker hub), run:

```
docker run -v /:/mnt --rm -iy debian:10-slim chroot /mnt bash
```

## (3/5) MAX CAN BECOME SYSADMIN WITH A POOR THOUGHT ENCRYPTION, THEN EXPLOIT A SUID PROGRAM

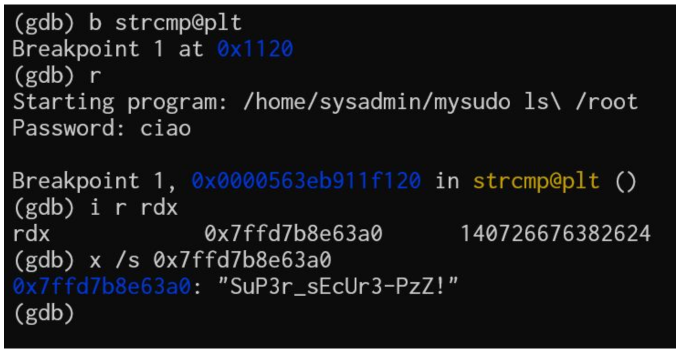
Not that realistic, but sometimes users have very little awareness of what is secure enough to store credentials and sensitive info and what is not. Plus, the SUID binary, that isn't realistic too (we hope), it is a lot of fun.

### Configuration

We implemented a vigenere encoder in c++, compiled and generated test files, we wrote a custom SUID program named mysudo, code is here: [https://pastebin.com/tBhBN9M8](https://pastebin.com/tBhBN9M8). After setting all the permission bits, it worked with intended flaws.

### How to exploit

Max has a backup folder in his home directory, containing a c++ program that implements a weak encoding algorithm, the corresponding binary, a test input and a test output and the password of sysadmin (obviously encoded with that program). The algorithm is the vigenere one, so the password is easily obtained by using the test files (clear text as decryption key for the ciphertext). As sysadmin, there's a custom c program (`mysudo`). Obfuscated so that `strings` and `strace` are useless. The only way is `gdb`:



```
(gdb) b strcmp@plt
Breakpoint 1 at 0x1120
(gdb) r
Starting program: /home/sysadmin/mysudo ls\ /root
Password: ciao

Breakpoint 1, 0x0000563eb911f120 in strcmp@plt ()
(gdb) i r rdx
rdx            0x7ffd7b8e63a0      140726676382624
(gdb) x /s 0x7ffd7b8e63a0
0x7ffd7b8e63a0: "SuP3r_sEcUr3-PzZ!"
(gdb)
```

## (4/5) EXPLOIT SCRIPT THAT DOESN'T USE ABSOLUTE PATHS FOR THE PROGRAMS THAT ARE CALLED

Same as 1.

### Configuration

We wrote a custom script that calls system utilities with relative paths, we gave webadmin the possibility to run that script as sudo without password:

```
webadmin ALL=(root) NOPASSWD:/usr/bin/machine-status
```

### How to exploit

The user can run a custom script (`/usr/bin/machine-status`), that is readable. Reading it, you can see that it calls utilities with relative paths, so it is possible to write your own version of `fdisk` (for example) and export its path at the beginning of PATH, so that it will be executed instead of the real binary as root.

## (5/5) all

Realistic because some tasks are automated in a system, but sometimes the ways to do it are poorly implemented.

### Configuration

We set up a cron job executed by root that executes all the files in a user-writable directory: Here's the script: https://pastebin.com/aG1aX6Q0. To make the cron job run every minute, we edited the crontab as follows:

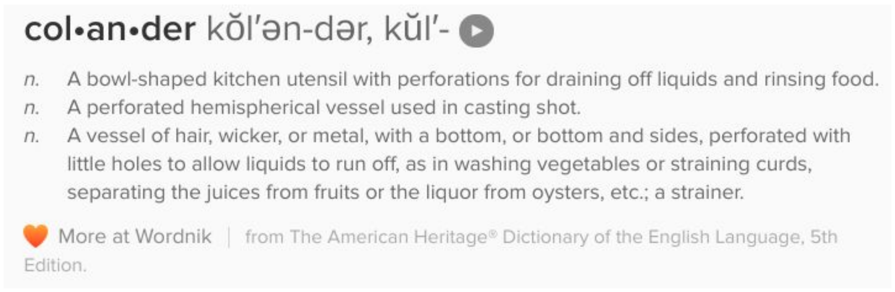
```
* * * * * /etc/cron-custom/maintenance.sh
```

### How to exploit

With enumeration it's possible to find `/etc/cron-custom/maintenance.sh`. That is a script that executes all the `*.sh` files placed in `/var/custom`. So, writing a script and placing it there should do something. (actually a cron job  that runs every minute will trigger its execution).

## LAST COMMENTS

The machine we designed it’s not the hardest one to exploit by any means, we wanted to do something funny and challenging at the same time. We learnt a lot by configuring the VM to be weak, and sometimes it was counterintuitively harder to make services vulnerable. We had fun doing this. That’s the type of practical we were looking for.



**col•an•der** kŏl′ən-dər, kŭl′-

| | |
|---|---|
| *n.* | A bowl-shaped kitchen utensil with perforations for draining off liquids and rinsing food. |
| *n.* | A perforated hemispherical vessel used in casting shot. |
| *n.* | A vessel of hair, wicker, or metal, with a bottom, or bottom and sides, perforated with little holes to allow liquids to run off, as in washing vegetables or straining curds, separating the juices from fruits or the liquor from oysters, etc.; a strainer. |

More at Wordnik | from The American Heritage® Dictionary of the English Language, 5th Edition.