Attacks, attack detection and attack analysis in Cloud environments

In Project 2, you will use **libvmi** and **volatility**, among other tools, with the following goals:

- Use VMI to detect and analyse a malware attack with a stealthy rootkit
- Analyse the interactions of a remote attacker with sshd
- Explore the potential of (mis-)using VMI for privilege escalation (root privileges in the target VM)

Setup: On our OpenNebula infrastructure you can create two VM images (you can continue using the VMs you created in Lab 2.2): **Wordpress (PVM)**, a virtual machine running a Wordpress installation, and **Monitor (MVM)**. *Please create only one pair of VMs per team.* Make note of the internal names of the virtual machines (**one-*xxx***) and their IP addresses (**192.168.13.*yyy***).

Tasks:

1. Malware detection using VMI with volatility

Read full task description first!

(a) Implement a simple intrusion detection and analysis tool that can extract information about
- running processes
- loaded kernel modules
- hidden kernel modules (as identified by the volatility plugin linux_hidden_modules)
- and network connections

using virtual machine introspection.
The tool should extract the information every *RUNINTERVAL* seconds (configurable), and display new and missing entries compared to the previous run (no output is expected on the first interval. Only completely new or no-longer-existing processes/kernel modules/network connections should be shown).

Note: You decide how to implement this tool. For example, you probably can solve this task with a sophisticated bash script, but you might also implement, e.g., a Java application that makes use of volatility, or a python program, or your own volatility plug-in.

Not all differences are malicious attacks. For example, if an administrator logs in into the wordpress VM and runs the command “top”, and some wordpress user accesses the wordpress server using a web browser, causing the apache web server to launch additional worker process, the output of your tool should be more or like this:

```
Processes:
+ 14270  0      0      sshd: root [priv]
+ 14271  113    65534  sshd: root [net]
+ 14272  0      0      /bin/bash
+ 14273  0      0      /usr/bin/top
+ 13918  33     33     /usr/sbin/apache2 -k start
+ 13917  33     33     /usr/sbin/apache2 -k start

Kernel modules:

Hidden kernel modules:

Network connections:
+ TCP 192.168.13.55:22 10.0.13.10:22732 ESTABLISHED  sshd/14270
```

(b) After preparing everything, you can run the commands

```
mkdir /attack
mount 192.168.13.19:srv /attack
touch /attack/192.168.13.yyy
```

on your **Monitor VM** to initiate a malware attack against your **Wordpress VW** (replace *yyy* with your Wordpress IP).

The file you created with "`touch`" will be deleted automatically after starting the attack, and after finishing the attack, we'll add the IP address to the end of the file `/attack/log.txt`

Use your tool developed in part (a) to display the difference between before and after the attack. Based on the results, explain what the malware appears to be doing. Note that you can attack your Wordpress machine only once. After that you have to shut it down and restart it (using OpenNebula). Using the "reboot" command in your VM will not work correctly – the network device may not get correctly initialized after reboot.

(c) Compare the results obtained with your VMI tool of part (a) with the in-guest results obtained by running the commands "`ps aux`", "`lsmod`" and "`netstat -tap`" as root user in your Wordpress VM after the attack? What artefacts of the attacks are visible to in-guest tools? Are there "hidden" changes to the system? Explain the results!
Note: It is sufficient to do this step manually. You may gain two additional bonus points by extending your IDS tool with a mechanisms to automatically collect the in-guest view of those commands and display the differences.

Store your detection tool in the folder **`ids`**. You should be able to run the tool as
`ids/vmidet <one-xxx> <RUNINTERVAL>`
Also add a Makefile in the `ids` folder if your tool requires compiling (e.g. if it is implemented in C or Java).

2. Virtual machine manipulation with libvmi

Using libvmi (or pyvmi) you can (besides reading) also write to guest VM memory (writing is not supported directly by volatility). You can find examples for using libvmi and pyvmi at:
https://github.com/libvmi/libvmi/blob/master/examples/process-list.c
https://github.com/libvmi/libvmi/blob/master/tools/pyvmi/README

Write a C or Python program that takes two arguments on the command line (name, pid) and changes the credentials of the process with PID <*pid*> in your wordpress VM <*name*> to root privileges. (Note: The credentials data structure is referenced by `task_struct` and shared among multiple processes. Directly changing the user-id value in the credentials structure will have an impact on multiple processes. This is **not** acceptable for this project. The better approach is to replace the pointer to the credentials data structure of process <pid> with the address of the credentials data structure of a process that already has root privileges, such as the `init` process).

Store your program "**uidchanger.py**" or "**uidchanger.c**" with all required additional files (e.g., header files, as well as a Makefile if you use C) in folder **`vmi`**.

Expected output running "ps aufx" command inside the wordpress VM, before:

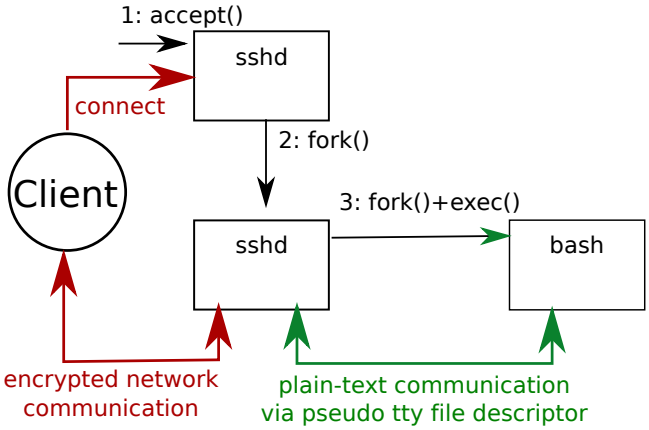
```
<previous processes>
tester   2008  0.0  0.1   8456   736 pts/1    S+   04:17   0:00   ping 8.8.8.8
<next processes>
```

After executing **uidchanger**:

```
<previous processes>
root   2008  0.0  0.1   8456   736 pts/1    S+   04:17   0:00   ping 8.8.8.8
<next processes>
```

3. System observation with system call tracing

Assume you noticed that there is some suspicious interaction with the sshd on your wordpress VM, and you want to investigate further what the ssh user (or attacker) is doing. Unfortunately all network interaction with sshd is encrypted, so you cannot observe the commands at the network level. You can, however, avoid this problem by using system call tracing.



This figure explains the interactions when a remote user accesses a system using ssh. For each user, sshd will create two processes, a copy of `sshd` and an instance of the interactive shall (`bash`). While the communication between remote user (client) and `sshd` is encrypted, the local interaction between `sshd` and `bash` is not. Thus, you can use system call tracing using libvmtrace to extract the input and output of the remote shell using VMI.

- First, you need to define some mechanism to detect the creation of a new ssh session (for example using network monitoring, using periodically checking the process list, or by tracing fork/exec system calls)
- Second, for each ssh session, you should use system call tracing of read/write system calls on file descriptors that are used for inter-process communication between sshd and bash (sshd receives encrypted user commands, decrypts them, and passes them to a child bash processes using write system calls on a pseudo-tty)
- For each sshd process, you should store the recorded information in some appropriate file format (one file for each client) that is easy to view for a (human) investigator.

Store all relevant things in the folder **`syscall`** of your repository.

Assume you have created a user "test" with password "testpass" on your wordpress VM, and a user connects to your VM using that account, types in the commands "echo "Hello World"", "ls /etc/pass*", and "exit", the human-readable output should be as follows (user input lines should be shown with prefix "> " , output lines without additional prefix):

```
------------
New ssh connection from <ip:port>:
> echo "Hello World"
Hello World
> ls /etc/pass*
passwd
> exit
------------
```

Final notes:

Each team member must be an expert in at least one of the three tasks and explain that task in the discussion.

Bonus question (this part is not mandatory, but you gain up to 2 additional bonus points that can be used to compensate other deficiencies)

VMI-based analysis depends on some assumptions made on the guest system, and the robustness of VMI tools differs a lot between existing tools.

- Put yourself into the role of an attacker that has gained full control (root) on the wordpress instance. What can you do in order to break the VMI-approach used in Part 1, 2, or 3 (choose one of them)? Note: "breaking" does not have to be stealthy, it is sufficient if the analysis tool fails by crashing or producing garbled output)?
- **Describe** one useful countermeasure against this attack.

Documentation:

Use the git repository for collaboration between team members, documentation, and submission of your final version. For that purpose, use the automatically created shared project

https://gandra.fim.uni-passau.de/csec_git/csec_exercises/group_<XX>_project_2

Include all your documentation in a single file **doc/project2.pdf** (or **`.txt`**/**`.md`**) of your git repository. The documentation of Project 2 will be the main basis for the evaluation, so make sure to include all relevant information! Include your group number, and names and student numbers of all **active** group members at the beginning of the documentation.

For submitting the final version, create and push an annotated **tag** “final” to the repository.

Important dates:

2019-07-16 11:59CET	Submission deadline
2019-07-18	Project discussions in the lab

Project discussions: Approx. 15 minutes interview on how you solved the tasks.

See Stud.IP for additional documentation and hints. Also please use the Stud.IP forum for discussing problems that you might run into.