Forensics Project Report
P3 - Volatility3 poisoning

Contributors: Marc Veysseyre, Sirshak Sarkar

Problem Statement:

Volatility 3 uses a signature to detect the correct radix-tree to start the analysis of a Linux dump. The signature ("swapper\0") is the name of the idle process and is contained in a field of its task_struct. What happens if the signature is overwritten by a malicious kernel module? The system continue to run correctly or crash? Volatility is able to continue the analysis? What happens if multiple process have this signature?

Is Volatility able to distinguish them?

This project had two parts:

Part 1

Take a memory dump from a linux machine and then modify it by overwriting the "swapper\0" string and then check the behaviour of Volatility3.

Additionally overwrite other processes with this swapper signature and then check if Volatility 3 is able to distinguish them.

Part 2

Develop a simple kernel module that modifies the task_struct of swapper\0 and check the stability of the image. Then create also other user-space program with the name equal to the signature and check the system behavior.

What we used:
Virtual Machines - Debian 12

IDE - Visual Studio Code
Tools and Frameworks - Volatility 3 Framework, dwarf2json, LiME

Understanding how task_struct works

In the linux operating system, there are data structures that represents a process object which is called a process control block PCB. The task_struct is the PCB in linux and is also the TCB meaning the Thread Control block. It is a data structure written in C which basically holds all information for a given process which our kernel requires to understand how to process information related to that process. Usually the most important information held by task_struct for a given process is the priority , process ID , parent process ID, list of open resources, memory space range information and namespace information. Supposedly the task_Struct can be accessed by using a pointer to the structure. In C we should be able to use a macro current to access this for the current process.

There is also another way where we can access task_struct, basically every process is in a stack in linux and at the top we have a structure called thread_info which itself has a pointer to the task_struct structure. On this link: c - How does the kernel use task_struct? - Stack Overflow it says we can "mask " the first 13 bits of the stack pointer to get the pointer to thread_info and then once we have the pointer inside thread_info we can use the current process macro which gives us the pointer to the task struct().

This may help: c - How programmatically get Linux process's stack start and end address? - Stack Overflow

List of data fields that the Linux kernel uses from task_struct

There are essentially a few categories of data fields:

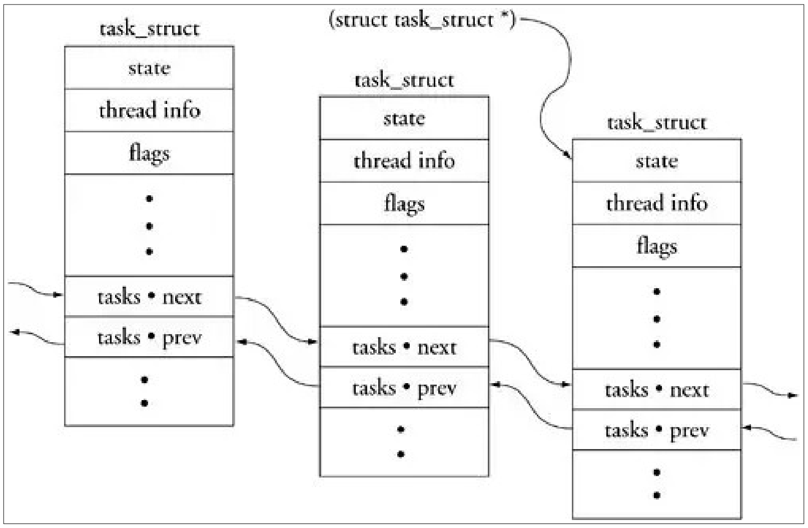
- **Process State and Identification**
- **Scheduling and Priority**
- **CPU Affinity and Allowance**
- **Process Relationships**
- **Process Exit Information**
- **Process Memory Management**
- **Signals and Notifications**
- **Resource and Accounting**
- **Files, Namespaces, and IO**
- **Task Context and Execution**
- **Auditing, Security, and Control**
- **Timers and Slacks**
- **Kernel-Specific Data**

There are some links which have some good explanations for many data fields:

Link: *https://linux-concepts.blogspot.com/2018/02/explanation-of-struct-taskstruct.html*
Source code: *https://docs.huihoo.com/doxygen/linux/kernel/3.7/structtask__struct.html*
Source code: *https://elixir.bootlin.com/linux/v6.2-rc1/source/include/linux/sched.h#L737*

The swapper process and signature

The task_struct function is a crucial component of the Linux kernel, and understanding its operation is essential for modifying the signature of the task structure for the first process to spawn, known as the swapper. This process, with Process ID (PID) 0, serves as its own parent, with its parent pointer self-referencing. The swapper process functions as the idle task, executing when the CPU has no other processes to run and playing a vital role in process scheduling.



The signature or name "swapper/0" or "swapper" for the idle task is stored in the 'comm' field of the task_struct data structure. This 'comm' field is a character array that holds the name of the process, and it is this field that requires modification. For the swapper process (PID 0), the 'comm' field is initialized to "swapper/0" for CPU 0, and "swapper/n" for the nth CPU in symmetric multiprocessing (SMP) systems.

It is important to note that checking if the current PID is 0 is a common method to identify the swapper process in kernel code. The 'comm' field for the swapper process in the task_struct is 16 characters long, including the null terminator. For the swapper process, it contains the string "swapper/0" followed by null characters to fill the remaining space. There are no additional characters within the comm field itself

Modifying this field will alter the signature of the swapper process.

Setting Up

We used a Debian 12 Virtual Machine inside VmWare to get the memory dump. This was done using dwarf2json. The memory dump required 2 components the System Map and the linux kernel vmlinux kernel symbol file for the target memory dump.

Debian stores the kernel images and the corresponding System.map and vmlinux files in its repositories. So we can just download the appropriate kernel package that matches our kernel version.

sudo apt update

sudo apt install linux-headers-6.1.0-21-amd64 linux-image-6.1.0-21-amd64-dbg

Next we used dwarf2json to create the profile using the System map and the vmlinux file from the debug boot directory.

sudo apt install golang-go

git clone https://github.com/volatilityfoundation/dwarf2json.git

cd dwarf2json

go build

sudo mv dwarf2json /usr/local/bin/

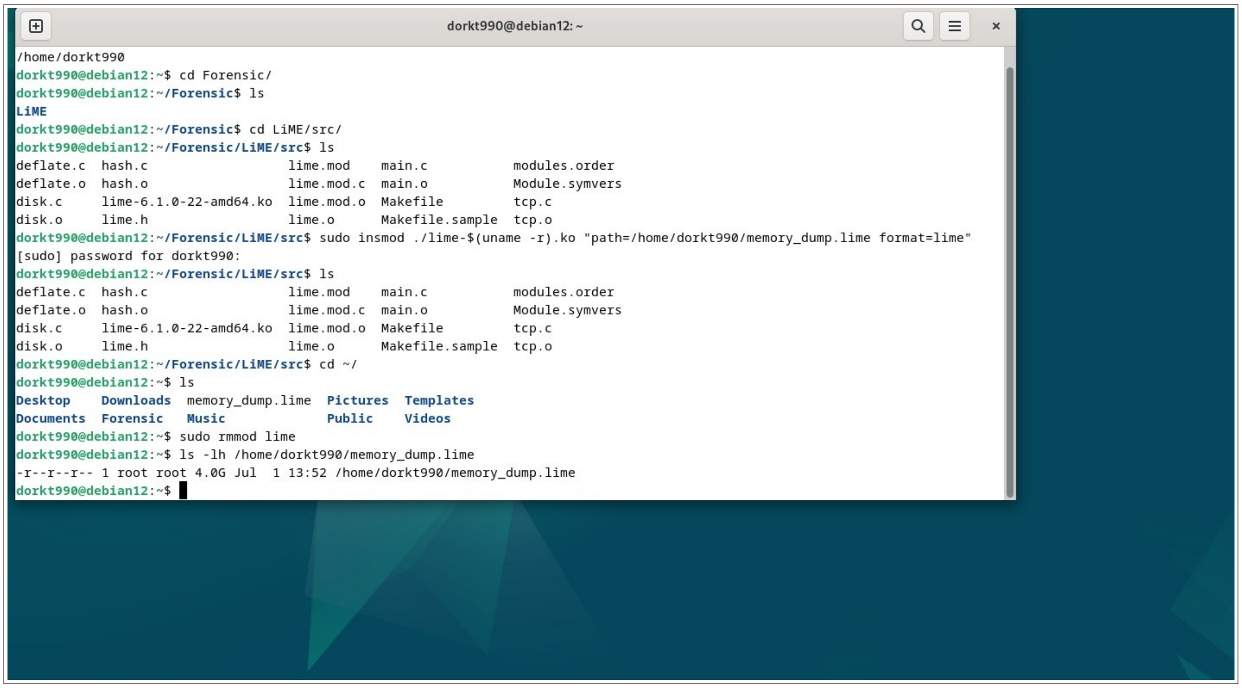
sudo dwarf2json linux --elf /usr/lib/debug/boot/vmlinux-$(uname -r) --system-map /usr/lib/debug/boot/System.map-$(uname -r) > debian.json

Additionally we also took a system memory dump using LiMe module. Ofcourse we had to make it first.

cd ~/Forensic/LiME/src

make clean

sudo make



And then we simply use it as a kernel module to take the memory dump.

> ***sudo insmod ./lime-$(uname -r).ko "path=/home/dorkt990/memory_dump.lime format=lime"***

After this we have a memory dump ready to use after we build our profile for volatility3.

Building the profile for our target kernel

Volatility 3 uses a slightly different system than Volatility 2 so the profiles need to be made with the system map and the actual kernel debug headers which we downloaded ofcourse.

After these two files are available we can use dwarf2json to build the profile which is essentially a gigantic json file full of important data-metadata associations for symbols that can be found in the memory dump.

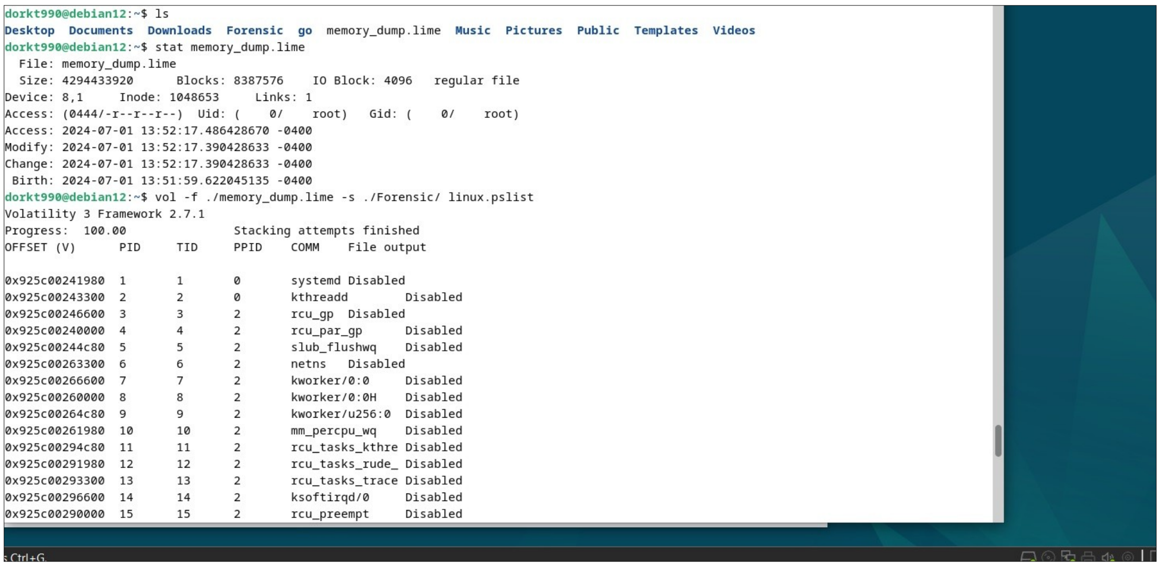
```
dorkt990@debian12:~/Forensic/volatility3$ dwarf2json -h
Usage: dwarf2json COMMAND

A tool for generating intermediate symbol file (ISF)

Commands:
  linux  generate ISF for Linux analysis
  mac    generate ISF for macOS analysis

dorkt990@debian12:~/Forensic/volatility3$ cd ..
dorkt990@debian12:~/Forensic$ sudo dwarf2json linux --elf /usr/lib/debug/boot/vmlinux-$(uname -r) --system-map /usr/lib/debug/boot/System.
map-$(uname -r) > debian.json
dorkt990@debian12:~/Forensic$ ls
debian.json  dwarf2json  LiME  volatility3
dorkt990@debian12:~/Forensic$ stat debian.json
  File: debian.json
  Size: 37871684        Blocks: 73984      IO Block: 4096   regular file
Device: 8,1     Inode: 1048878     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/dorkt990)   Gid: ( 1000/dorkt990)
Access: 2024-07-01 14:00:44.115792473 -0400
Modify: 2024-07-01 14:01:25.231639106 -0400
Change: 2024-07-01 14:01:25.231639106 -0400
 Birth: 2024-07-01 14:00:44.115792473 -0400
dorkt990@debian12:~/Forensic$
```

The command used is:

sudo dwarf2json linux --elf /usr/lib/debug/boot/vmlinux-$(uname -r) --system-map /usr/lib/debug/boot/System.map-$(uname -r) > debian.json

Using the profile to get our first look at the memory dump

Now we can use volatility 3 to check all processes running with linux.pslist . It is important to note that the profile path has to be supplied and the memory dump path as well.



We can see here that the PID 0 doesn’t exist. This is because it is ignored by volatility source code, the process just spawns and allows other processes to take over and is therefore quite redundant after it spawns. So we can’t actually see the PPID of PID 1.

Using our Notebook for streamlining interactions

We made a jupyter notebook which is fed with our kernel module and the commands to identify, modify the swapper and other processes and generate modified memory dumps with all our commands and outputs.

First we focus on setting up, we used code to run our notebook. And then set up the correct path settings in the first part of the notebook. it may ask you to install ipyknernel if you used code.

sudo apt install software-properties-common apt-transport-https wget

wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main"

sudo apt update

sudo apt install code

cd ~/

git clone https://gitlab.eurecom.fr/veysseyr/forensics_pr3.git

cd forensics_pr3

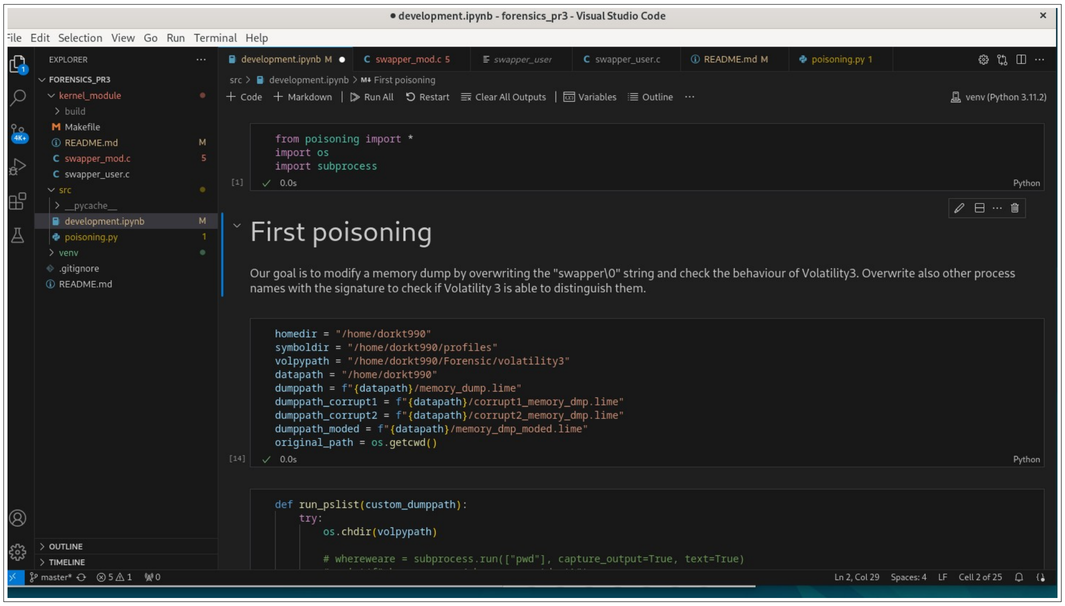
sudo apt install python3-venv python3-pip

source venv/bin/activate

we also installed volatility 3 for this new environment with

pip install git+https://github.com/volatilityfoundation/volatility3.git

We recommend setting up a separate folder for profiles in the home directory and placing the json profile. The first part of the notebook must be setup for it to work.



We created a custom python script called poisoning.py which used logic from the volatility API and source code. Below is a general summary of the basic functionalities that it allows us to have:

File Manipulation:

- It provides functions to duplicate memory dump files.
- It includes methods to search for specific patterns within large files efficiently.

Memory Dump Modification:

- The script can modify specific signatures within the memory dump, such as the "swapper" signature.
- It can insert new data into the dump file at specific locations without overwriting existing content.
- There's functionality to modify data at specific physical offsets in the dump file.

Process Manipulation:

- It includes a function to modify process names within the memory dump.

Address Translation:

- We have a function to convert virtual Linux addresses to physical addresses but it doesn't take into account ASLR or KASLR.

Traps and Testing:

There are "trap" functions that combine multiple operations, for example a test function is provided to verify the process name modification.

Finding the swapper process via signature

We first find the swapper function through it's signature

swapper(\/0|\x00\x00)\x00\x00\x00\x00\x00\x00.

Here is it important to note that the signature is in the task_struct of every process in the data field called COMM field.

The comm field in task_struct is a fixed-size character array of length TASK_COMM_LEN, which is defined as 16 in include/linux/sched.h.

For the swapper process (PID 0), this comm field is initialized to "swapper/0" (10 characters including null terminator).

Since the comm field has a fixed size of 16 bytes, and C strings require a null terminator ('\0' or 0x00), the remaining 6 bytes (15 characters) after "swapper/0" are likely filled with null characters (0x00).

This is a common practice in C to ensure that the string is properly null-terminated and does not overflow the fixed-size buffer.

While our search did not find any reason as to why it is like this, it is a logical conclusion based on the fixed size of the comm field, the requirement for null termination in C strings, and the initialization of the field with "swapper/0" for the swapper process.

Given this info we made a new signature:

new_signature = rb"swapLOL(\/0|\x00\x00)\x00\x00\x00\x00\x00\x00"

And then we can run our first command from the notebook to find and replace the swapper signature.

```
duplicate_dump(dumppath, dumppath_corrupt1)
modify_swapsign_in_dump(dumppath_corrupt1, swapper_signature, new_signature)
✓ 26.9s

Dump file duplicated: /home/dorkt990/corrupt1_memory_dmp.lime
Found swapper signature at offset: 729518624
Found swapper signature at offset: 1689476704
Found swapper signature at offset: 2664729248
Modified 3 occurrences of swapper signature.

run_pslist2(dumppath_corrupt1)
✓ 13.2s

cmd = vol -s /home/dorkt990/profiles -f /home/dorkt990/corrupt1_memory_dmp.lime linux.pslist.PsList
Volatility 3 Framework 2.7.1
Progress:  100.00               Stacking attempts finished

OFFSET (V)      PID     TID     PPID    COMM    File output
```

We essentially search for and modify the swapperr signatures in large memory dump file. The search_in_chunks function efficiently searches for a pattern in a file by reading it in chunks, which is memory-efficient for large files. It yields the positions of matches found.

Then the duplicate_dump function creates a copy of the original dump file for safe modification. The modify_swapsign_in_dump function uses search_in_chunks to find all occurrences of a specific "swapper signature" in the dump file, then replaces each occurrence with a new signature of the same length. It first checks that the new signature is the same length as the old one to maintain file structure integrity. This approach allows for targeted modifications of memory dumps which is what we want!

We also see that after the modification if we run pslist the doubly linked process list structure which is parsed by volatility 3 doesn’t work and since the task_struct of the swapper has been modified volatility 3 cannot work properly.

Modifying the signature destroys volatility3’s ability to work on memory dumps!

Interestingly after we modify the signature the swapper and we can see the swapper signature at 3 different memory offsets. This is quite weird. We are unsure why this is the case. After testing on several memory dumps on different (but same Debian version) VM’s we saw it was quite volatile.

Modifying Other Processes and signatures

Furthermore we select a process which has the same signature (process name in this case) length. This is quite important we cannot replace the signature length meaning reduce or increase it, so we selected the

process_signature = rb"slub_flushwq"

We run the modification as before and we find the signature for the process at several offsets.

We can see that after the modification in the pslist the process signature has been replace with swapper/0. But it doesn't affect the stability of the memory dump or volatility3.

```
run_pslist2(dumppath_corrupt2)
✓ 8.6s

cmd = vol -s /home/dorkt990/profiles -f /home/dorkt990/corrupt2_memory_dmp.lime linux.pslist.PsList
Volatility 3 Framework 2.7.1
Progress:  100.00               Stacking attempts finished

OFFSET (V)      PID     TID     PPID    COMM    File output

0x925c00241980  1       1       0       systemd Disabled
0x925c00243300  2       2       0       kthreadd        Disabled
0x925c00246600  3       3       2       rcu_gp  Disabled
0x925c00240000  4       4       2       rcu_par_gp      Disabled
0x925c00244c80  5       5       2       swapper(\/0)    Disabled
0x925c00263300  6       6       2       netns   Disabled
0x925c00266600  7       7       2       kworker/0:0     Disabled
0x925c00260000  8       8       2       kworker/0:0H    Disabled
0x925c00264c80  9       9       2       kworker/u256:0  Disabled
0x925c00261980  10      10      2       mm_percpu_wq    Disabled
0x925c00294c80  11      11      2       rcu_tasks_kthre Disabled
```

We can now see that PID 5 has been modified. And if we repeat this for another signature by exchanging the swapper signature with another signature say for the process called:

migration/0

we can see that, the signature will be found in the memory by the script. But it will destroy the process double linked list used by volatility3 and no processes will be shown after pslist is run.

```
process_signature_2 = rb"migration/0"

modify_extension_in_dump(dumppath_corrupt2, process_signature_2, swapper_signature)
```
[25] ✓ 13.8s

```
Modified occurrence at offset: 3223434400 with new data.
Modified 1 occurrences of the wanted signature.
```

```
run_pslist2(dumppath_corrupt2)
```
[31] ✓ 22.4s

```
cmd = vol -s /home/dorkt990/profiles -f /home/dorkt990/corrupt2_memory_dmp.lime linux.pslist.PsList
Volatility 3 Framework 2.7.1
Progress:  100.00               Stacking attempts finished

OFFSET (V)      PID     TID     PPID    COMM    File output
```

Using our Kernel Module for Part 2

We developed a simple kernel module that modifies the task_struct of swapper\0 and then take a fresh memory dump of the modified kernel.

We have two main programs which are built using a makefile. This has to be done manually. The main modification program is essentially a user-space program interacting with swapper device.

This Linux kernel module (swapper_mod.c) creates a character device and modifies the task structure of the swapper process (PID 0). The module includes the following key features:

- **Device Creation**: It dynamically allocates a major number and creates a character device named "swapper_device".
- **File Operations**: Implements basic file operations (open, read, write, release) for the device, though they only print debug messages and don't perform substantial actions.

Task Structure Modification: The modify_task_struct function loops through all processes to find the swapper process (PID 0) and modifies its priority as an example.

Initialization and Cleanup: The module uses a standard init and exit functions clean up the device after use.

Kernel Interaction: It interacts with various kernel subsystems, including process management, device management, and the virtual filesystem.

Kernel Interaction and Task_struct Modification

The module directly modifies the task_struct of the swapper process (PID 0). Specifically it uses the for_each_process macro to iterate through all processes in the system.

When it finds the swapper process (identified by PID 0), it modifies the prio field of its task_struct.

The priority is set to 20, which is an arbitrary value and not recommended in practice.

If we expand on our interaction with kernel subsystems we can summarize them as:

Character Device Subsystem: It creates a character device named "swapper_device" using functions like alloc_chrdev_region, cdev_init, cdev_add, and device_create.

Process Management: It uses the for_each_process macro to iterate through all processes and accesses the task_struct directly.

Sysfs: It creates a device class using class_create and a device node using device_create, which interact with the sysfs filesystem.

Kernel Logging: It uses printk to log messages to the kernel log buffer.

Module System: It uses module_init and module_exit macros to define initialization and cleanup functions for the module.

Understanding the Kernel Module Interactions

Now we insert the module after we build it and use dmesg to see the kernel logs.

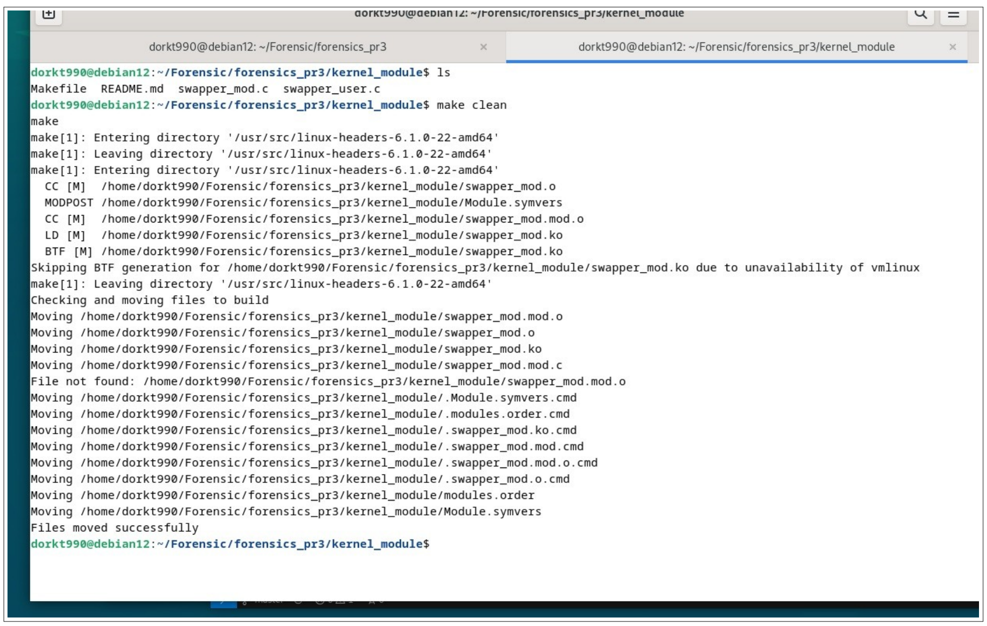
sudo rmmod swapper_mod

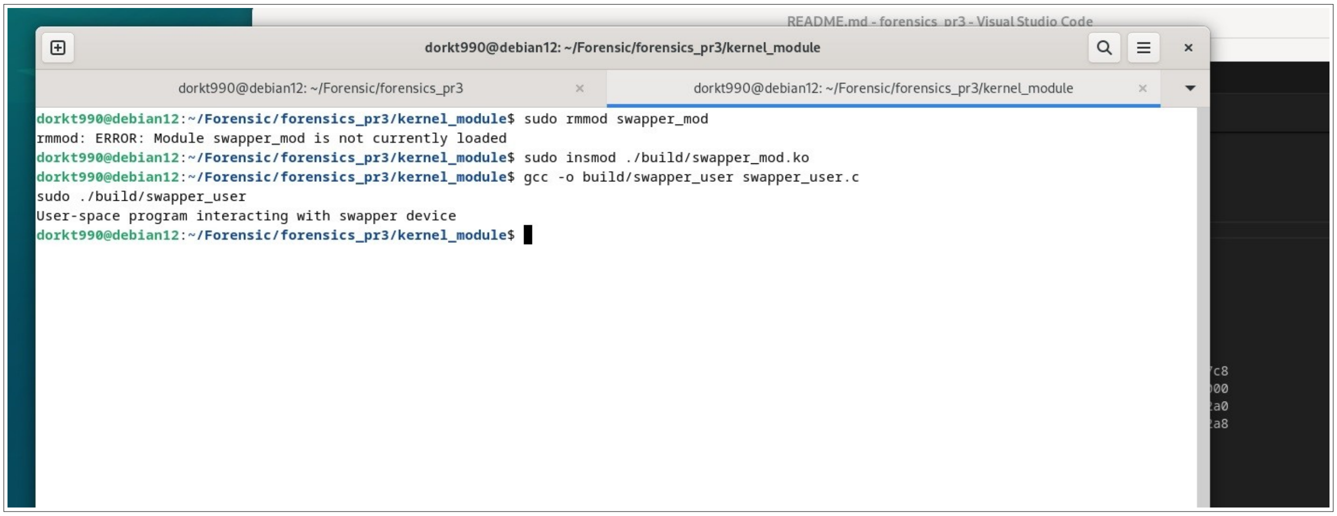
sudo insmod ./build/swapper_mod.ko

gcc -o build/swapper_user swapper_user.c

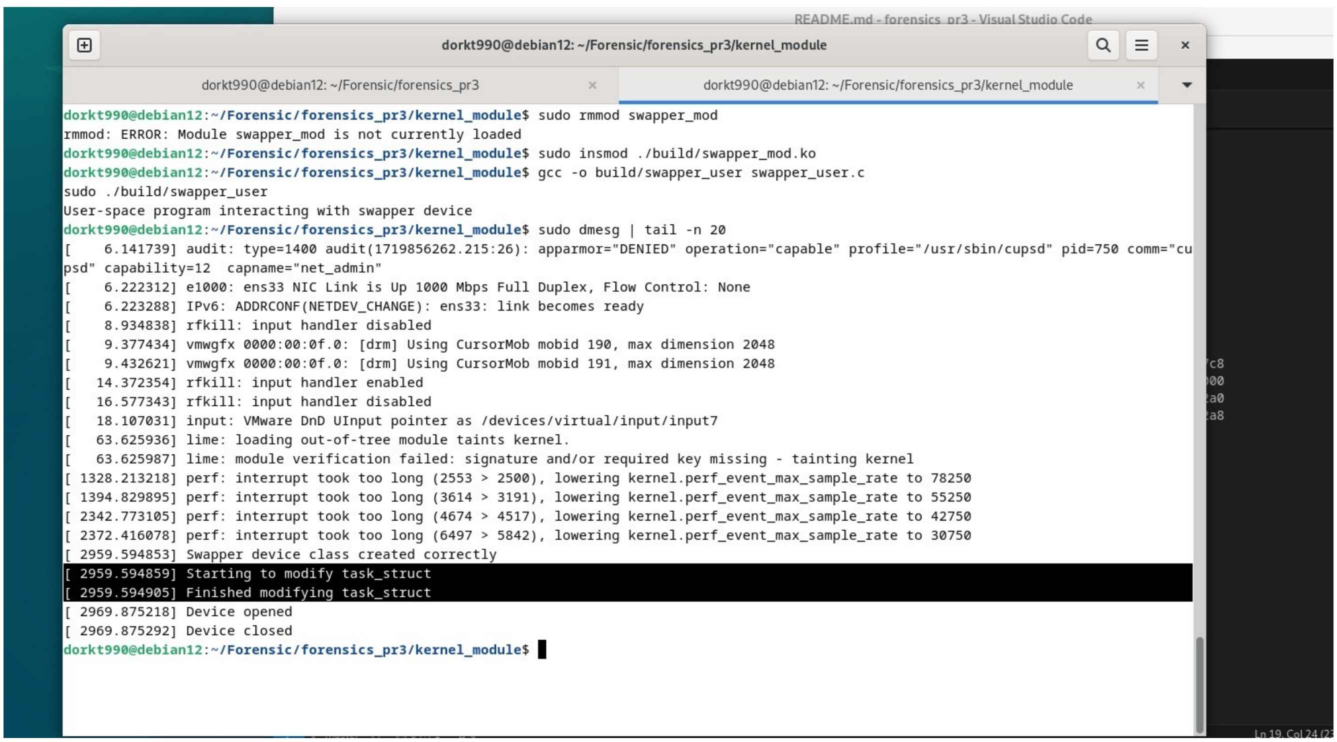
sudo ./build/swapper_user

sudo dmesg | tail -n 20





We can see the live guest VM kernel swapper being modified in the logs.



Here are some logs in text which show exactly how the swapper interacts with the kernel:

dorkt990@debian12:~/Forensic/LiME/src$ sudo dmesg | tail -n 20
[8.934838] rfkill: input handler disabled
[9.377434] vmwgfx 0000:00:0f.0: [drm] Using CursorMob mobid 190, max dimension 2048
[9.432621] vmwgfx 0000:00:0f.0: [drm] Using CursorMob mobid 191, max dimension 2048
[14.372354] rfkill: input handler enabled

```
[   16.577343] rfkill: input handler disabled
[   18.107031] input: VMware DnD UInput pointer as /devices/virtual/input/input7
[   63.625936] lime: loading out-of-tree module taints kernel.
[   63.625987] lime: module verification failed: signature and/or required key missing - tainting kernel
[ 1328.213218] perf: interrupt took too long (2553 > 2500), lowering kernel.perf_event_max_sample_rate to 78250
[ 1394.829895] perf: interrupt took too long (3614 > 3191), lowering kernel.perf_event_max_sample_rate to 55250
[ 2342.773105] perf: interrupt took too long (4674 > 4517), lowering kernel.perf_event_max_sample_rate to 42750
[ 2372.416078] perf: interrupt took too long (6497 > 5842), lowering kernel.perf_event_max_sample_rate to 30750
[ 2959.594853] Swapper device class created correctly
[ 2959.594859] Starting to modify task_struct
[ 2959.594905] Finished modifying task_struct
[ 2969.875218] Device opened
[ 2969.875292] Device closed
[ 3088.656852] perf: interrupt took too long (8317 > 8121), lowering kernel.perf_event_max_sample_rate to 24000
[ 3482.484043] lime: unknown parameter 'verbose' ignored
[ 4120.962126] lime: unknown parameter 'debug' ignored
```

Here are some actual lines we can understand what happened

[2959.594853] Swapper device class created correctly

This indicates that the module has been loaded successfully and the device class for the"swapper_device" has been created.

[2959.594859] Starting to modify task_struct
[2959.594905] Finished modifying task_struct

These two lines show that our modify_task_struct() function was called and completed its execution. It attempted to modify the swapper process's task_struct.

[2969.875218] Device opened
[2969.875292] Device closed

These lines indicate that a user-space program opened and then closed our device. This triggered our device_open() and device_release() functions.

Running volatility 3 on the modified memory dump

Now we after loading the kernel module we take a fresh memory dump using LiME.

cd ../../LiME/src

output_file="../../data/memory_dmp_moded.lime"

sudo insmod lime-6.1.0-21-amd64.ko "path=$output_file format=lime"

#if the above doesnt work then use absolute path like

sudo insmod lime-6.1.0-22-amd64.ko path=/home/dorkt990/memory_dmp_moded.lime format=lime

sudo rmmod lime

sudo chown dadmin:dadmin $output_file

chmod 666 $output_file

We have to make sure the modified file is user accessible. So some ownership and file permission changes are required before we can check the final pslist to see if our swapper task_struct modifications had any effect on volatility3's ability to run correctly.

```
dorkt990@debian12:~/Forensic/LiME/src$ sudo rmmod lime
dorkt990@debian12:~/Forensic/LiME/src$ sudo chown dorkt990:dorkt990 ~/memory_dump.lime
dorkt990@debian12:~/Forensic/LiME/src$ sudo chmod 666  ~/memory_dump.lime
dorkt990@debian12:~/Forensic/LiME/src$ 
```

Conclusions

Finally we go back to our notebook and run the process list command. **It works correctly.**

2.1 Test Volatility3 Stability on modified task_struct

write a simple kernel module that modifies the task_struct of swapper\0 and check the system stability. done ✅

```
[30]  run_pslist2(dumppath_moded)
      ✓ 7.9s
```

```
cmd = vol -s /home/dorkt990/profiles -f /home/dorkt990/memory_dmp_moded.lime linux.pslist.PsList
Volatility 3 Framework 2.7.1
Progress:  100.00               Stacking attempts finished

OFFSET (V)      PID     TID     PPID    COMM    File output

0x925c00241980  1       1       0       systemd Disabled
0x925c00243300  2       2       0       kthreadd        Disabled
0x925c00246600  3       3       2       rcu_gp  Disabled
0x925c00240000  4       4       2       rcu_par_gp      Disabled
0x925c00244c80  5       5       2       slub_flushwq    Disabled
0x925c00263300  6       6       2       netns   Disabled
0x925c00260000  8       8       2       kworker/0:0H    Disabled
0x925c00261980  10      10      2       mm_percpu_wq    Disabled
0x925c00294c80  11      11      2       rcu_tasks_kthre Disabled
0x925c00291980  12      12      2       rcu_tasks_rude_ Disabled
0x925c00293300  13      13      2       rcu_tasks_trace Disabled
0x925c00296600  14      14      2       ksoftirqd/0     Disabled
0x925c00290000  15      15      2       rcu_preempt     Disabled
0x925c0029cc80  16      16      2       migration/0     Disabled
0x925c00299980  17      17      2       kworker/0:1     Disabled
0x925c00a1b300  18      18      2       cpuhp/0 Disabled
0x925c00a3cc80  19      19      2       cpuhp/1 Disabled
0x925c00a39980  20      20      2       migration/1     Disabled
```

There are some reasons for this!

Here is the conclusion we reached:

system stability after swapper modification actually depends on what type of modifications you make on the swapper's task_struct, which is quite interesting, meaning some data fields in the structure can be modified and some are off limits.

For our case we decided to modify the task priority which was set to 20.

```c
static void modify_task_struct(void) {
    struct task_struct *task;

    printk(KERN_INFO "Starting to modify task_struct\n");

    // Loop through each process
    for_each_process(task) {
        if (task->pid == 0) {  // swapper has PID 0
            printk(KERN_INFO "Swapper task found: %s (PID: %d)\n", task->comm, task->pid);
            // Example modification: Change priority
            task->prio = 20; // Just as an example, not recommended!
            // but who cares
            printk(KERN_INFO "Modified swapper's priority\n");
            break;
        }
    }

    printk(KERN_INFO "Finished modifying task_struct\n");
}
```

So we can conclusively say that system stability will depend on what variables you modify in the task structure of the swapper and replacing any number of user space programs with the signature of the swapper may cause crashes as we see in some tests explained below. It may cause problems with program execution and system and data corruption.

But any change to the original swapper signature imapirs the integrity of the double linked list used to spawn all child processes which is why volatility3 shows nothing for the pslist.

Interestingly, if we run linux.check_syscall, linux.check_modules, on all signature modified memory dump files we get the same error:

```
scan
INFO     volatility3.framework.automagic: Detected a linux category plugin
INFO     volatility3.framework.automagic: Running automagic: ConstructionMagic
INFO     volatility3.framework.automagic: Running automagic: SymbolCacheMagic
INFO     volatility3.framework.automagic: Running automagic: LayerStacker
INFO     volatility3.schemas: Dependency for validation unavailable: jsonschema
INFO     volatility3.framework.automagic: Running automagic: SymbolFinder
INFO     volatility3.framework.automagic: Running automagic: LinuxSymbolFinder
INFO     volatility3.schemas: Dependency for validation unavailable: jsonschema
INFO     volatility3.framework.automagic: Running automagic: KernelModule

Table Address   Table Name      Index   Handler Address Handler Symbol


Volatility was unable to read a requested page:
Page error 0xffff82000320 in layer layer_name (Page Fault at entry 0x1f490001e430 in table page directory pointer)

        * Memory smear during acquisition (try re-acquiring if possible)
        * An intentionally invalid page lookup (operating system protection)
        * A bug in the plugin/volatility3 (re-run with -vvv and file a bug)

No further results will be produced
```

Page error 0xffff82000320 in layer layer_name (Page Fault at entry 0x1f490001e430 in table page directory pointer)

```
/olatility was unable to read a requested page:
Page error 0xffff82020744 in layer layer_name (Page Fault at entry 0x1f490001e430 in table page directory pointer)

        * Memory smear during acquisition (try re-acquiring if possible)
        * An intentionally invalid page lookup (operating system protection)
        * A bug in the plugin/volatility3 (re-run with -vvv and file a bug)

No further results will be produced
```

```
         ^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/objects/templates.py", line 96, in __call__
    return self.vol.object_class(
           ^^^^^^^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/objects/__init__.py", line 168, in __new__
    value = cls._unmarshall(context, data_format, object_info)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/objects/__init__.py", line 202, in _unmarshall
    data = context.layers.read(
           ^^^^^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/interfaces/layers.py", line 638, in read
    return self[layer].read(offset, length, pad)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/layers/linear.py", line 45, in read
    for offset, _, mapped_offset, mapped_length, layer in self.mapping(
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/layers/intel.py", line 295, in mapping
    for offset, size, mapped_offset, mapped_size, map_layer in self._mapping(
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/layers/intel.py", line 351, in _mapping
    chunk_offset, page_size, layer_name = self._translate(offset)
                                          ^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/layers/intel.py", line 155, in _translate
    entry, position = self._translate_entry(offset)
                      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framework/layers/intel.py", line 196, in _translate_entry
    raise exceptions.PagedInvalidAddressException(
/olatility3.framework.exceptions.PagedInvalidAddressException: Page Fault at entry 0x1f490001e430 in table page directory pointer
```

The page faults occur at different memory offsets, depending on what command was run. They are volatile, this clearly indicates system instability.

Now for the **final modified memory dump** , where we used the kernel module, volatility shows an **interesting log!**

We used linux.capabilities.

```
                ^^^^^^^^^^^^^^^^^^^^^^
 File "/home/dorkt990/Forensic/forensics_pr3/venv/lib/python3.11/site-packages/volatility3/framewor
   raise exceptions.VolatilityException(
'olatility3.framework.exceptions.VolatilityException: Unsupported kernel capabilities implementation

'olatility encountered an unexpected situation.


       * Please re-run using with -vvv and file a bug with the output
       * at https://github.com/volatilityfoundation/volatility3/issues

lo further results will be produced
```

So for this plugin, volatility 3 does not have a exception resolution for this class of errors, where the swapper was modified.

Volatility 3 works as expected for the kernel modified dump with the plugins, pslist, proc.Map, check_idt, check_syscall.

References

1. Explanation of "struct task_struct" (linux-concepts.blogspot.com)
2. c - How does the kernel use task_struct? - Stack Overflow
3. Workqueue — The Linux Kernel documentation
4. sched.h - include/linux/sched.h - Linux source code (v6.2-rc1) - Bootlin
5. How to generate a Volatility profile for a Linux system | Andrea Fortuna
6. GitHub - kd8bny/LiMEaide: A python application designed to remotely dump RAM of a Linux client and create a volatility profile for later analysis on your local host.
7. Linux memory analysis with Lime and Volatility – Blog by Jay Mutkawoa (Nitin) (tunnelix.com)
8. sched.h - include/linux/sched.h - Linux source code (v6.2.16) - Bootlin
9. printing the comm field of the `current` task_struct, Linux kernel - Stack Overflow
10. include/linux/sched.h (harvard.edu)
11. Linux · volatilityfoundation/volatility Wiki · GitHub
12. Volshell - A CLI tool for working with memory — Volatility 3 2.7.1 documentation
13. volatility3.plugins package — Volatility 3 2.7.0 documentation
14. Linux Kernel: task_struct Struct Reference (huihoo.com)
15. Processes — The Linux Kernel documentation (linux-kernel-labs.github.io)
16. coursys.sfu.ca