

---

This is the **published version** of the bachelor thesis:

Alonso, Anthony Michael; Martinez Garcia, Carles, dir. Ethical Hacking Framework for File System Hot-Swapping. 2022. (Enginyeria Informàtica)

---

This version is available at <https://ddd.uab.cat/record/280721>

under the terms of the  license

# Ethical Hacking Framework for File System Hot-Swapping

Anthony Michael Alonso

July 2, 2023

**Abstract**– This project aims to develop a framework that enables researchers to use our USB file system hot-swapping technique. This technique consists of changing the contents of any file on the USB device after the user has inserted it into the host computer and without removing the said device from the host, thus the hot-swapping aspect of the project. There are a multitude of legitimate, as well as malicious, use cases for this technique. While performing the hot swap is not difficult, this framework will facilitate investigation into this method by simplifying the process while providing an easy-to-use interface. This paper aims to educate the reader on the use cases and limitations, the minimally required equipment, the intricacies of this method, and finally, a walk-through of the framework and the PoC we have created.

**Keywords**– framework, hot-swap, investigation, limitations, proof-of-concept, technique, walk-through

**Resumen**– Este proyecto tiene como objetivo desarrollar un marco de trabajo que permita a los investigadores utilizar nuestra técnica de “hot-swap” de sistemas de archivos USB. Esta técnica consiste en cambiar el contenido de cualquier archivo en el dispositivo USB después de que el usuario lo haya insertado en su computadora, llevándose a cabo sin retirar dicho dispositivo de esta, de ahí el “hot-swap”, sustitución en caliente, del método. Hay una multitud de casos de uso legítimos, así como maliciosos, para esta técnica. Si bien realizar el “hot-swap” no es una tarea difícil, creemos que este marco de trabajo facilitará la investigación de este método al simplificar el proceso y proporcionar una interfaz fácil de usar. Este documento tiene como objetivo educar al lector sobre los casos de uso y limitaciones de esta técnica, el equipo mínimo requerido, las particularidades de este método y, finalmente, una guía paso a paso del marco de trabajo y un ejemplo de uso de este.

**Palabras clave**– guía paso a paso, hot-swap, investigacion, limitaciones, marco de trabajo, técnica

## 1 INTRODUCTION

THIS project’s research topic is to investigate and develop a framework that can exploit the ability to hot-swap the file contents of a USB device. The traditional method of making changes to a file on a USB device is to either make the modifications on a computer after inserting and opening the USB via a file browser, such as Finder on macOS or File Explorer on Windows, or re-

move the USB from the host, make the desired changes on another computer, and reinsert the USB into the target host. Our technique focuses on implementing this ability to change the contents of a specific file on a USB device, *without* removing the USB from the host. The ability to hot-swap contents of a USB opens up many exploitation possibilities, which can be benign to a host or malicious. Naturally, due to our curious nature and intent to provide value to the framework for security researchers, we will focus on the “malicious” angle of exploiting this technique.

### 1.1 Use Cases

Taking this approach, some of the ways this method could be used are as follows:

- Contact E-mail: [anthony@alonso.tv](mailto:anthony@alonso.tv)
- Specialization: Tecnologies de la Informació
- Project supervised by: Carles Martinez Garcia
- Curs 2022/23

⇒ Bypassing initial antivirus software scans of the USB.

- When a USB device is inserted into a computer (assuming it has an antivirus solution installed, such as BitDefender [1], and configured to do so), its contents are usually scanned to check if any malicious files are on the drive. If the files are deemed harmless, the user can interact with the USB device as expected, frequently involving read and write actions. The attentive reader will notice that it was stated that this occurs when a USB device is inserted. If we use our hot-swapping method, we can modify the contents of a file (or files) on the USB device without removing it from the host, bypassing the scan of the modified contents and permitting our “malicious” code to reside on the host.

⇒ Bypass the digital signature verification process [2] in real-time to execute malicious code.

- When a user attempts to run digitally signed software on a computer, the operating system usually has mechanisms that verify its signature. This process does its best to ensure that the software has not been tampered with, protecting the user from running unauthorized or malicious software on their machine. With this hot-swapping method, we can modify the contents of the software after it has been determined to be benign, thus bypassing the verification process and getting our code onto the target machine.

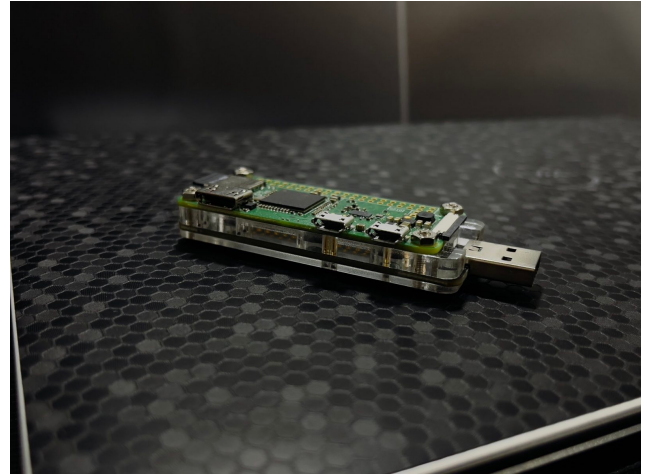
## 2 TECHNIQUE ANALYSIS

We have extensively researched how to execute the hot swap and understand what happens behind the scenes. After completing this project and gaining research experience, hopefully, we can get to the point at which we fully comprehend the processes that take place during the execution of this method. Until then, we will explain what we know to the best of our ability, aiming to point potential researchers in the right direction to appreciate the procedures during the hot swap.

### 2.1 Required Hardware & Software

It is not easy (more likely, impossible) to modify the contents of a regular USB flash drive, much less establish a connection to one while it is in use on another machine. However, we are in luck; we do have the ability to emulate mass storage devices (USB devices are a subset of these) on computers with a compatible Linux kernel by using the *g\_mass\_storage* kernel module [3]. We are going to use a Raspberry Pi Zero W [4] (referred to as the “Pi” in this document) to emulate a USB device via the *g\_mass\_storage* kernel module. Attaching a USB-A adapter [5] to the Pi will enable us to plug it into a computer to behave like a USB device. Our setup can be seen in Figure 1.

All tests and software were performed and developed on the Pi using Raspberry Pi OS Lite (32-bit) [6]. The *Lite* version differs from the full version in that it does not come



**Fig. 1:** Our Raspberry Pi Zero W with a USB-A adapter attached to it, enabling us to plug it into a computer easily.

with a fully-fledged GUI. The two reasons we chose to work with this version of Raspberry Pi OS are:

1. We do not need a GUI; we only need a terminal since we will connect to the Pi using SSH.
2. The Lite version is easier to set up from zero. If we use the official imaging toolkit [7] provided by Raspberry Pi, we can easily pre-configure handy options like connecting to a Wi-Fi network, enabling SSH, and so on, without being forced to follow the initial setup steps displayed in the full version of Raspberry Pi OS. This software is valuable because a special adapter is needed for a keyboard or mouse, or a compatible keyboard/mouse combination is required to perform these steps otherwise.

Once the OS is installed, there are 2 main configuration modifications that one should perform:

1. **Mandatory** Enable the modules *i2c-dev* and *dwc2* and the *dwc2* driver in their respective files. Our framework has a script that makes these changes if necessary. These adjustments enable the USB driver and the gadget mode for the Pi.
2. **Optional** One may want to enable a Wi-Fi hotspot on the Pi, depending on one’s environment. If this technique will be used outside of a familiar network and one wishes always to connect via SSH to the Pi whenever the Pi is turned on, this option will allow it. Our framework also includes a script to set this up. An important detail to remember is that the Pi will not have access to the Internet when emitting its own Wi-Fi network, and neither will any of the devices using the Pi’s Wi-Fi hotspot. The hotspot can easily be disabled, but depending on one’s environment, the Pi will not be accessible again until the hotspot is re-enabled.

After discussing the configuration of the Pi, we can discuss what occurs when this technique is executed.

### 2.2 Technique Internals

We will attempt to explain what happens under the following premise: *A USB device named **USB-1** has two files on*

it: *file1.txt* and *file2.txt*. The file whose contents we want to hot-swap, *file1.txt*, is now referred to as the “**target file**” in this section, if not by its original name.

### 2.2.1 Interacting with Files

When USB-1 is inserted into a computer, the interesting part occurs upon interaction with it and its contents. When a file is interacted with, let us say *file1.txt*, simply put, the file’s contents are *cached*<sup>1</sup> by storing them in the host’s memory. We can verify this in the figure below (2).

RES	PAGES	SIZE	FILE
0B	0	10.9K	file1.txt
0B	0	87.3K	file2.txt

(a) The file contents are not loaded in memory because we have not interacted with them, as we can see here.

RES	PAGES	SIZE	FILE
12K	3	10.9K	file1.txt
88K	22	87.3K	file2.txt

(b) The file contents are in memory now that we have interacted with them by simply opening them.

**Fig. 2:** Space in memory occupied by files in the USB. This view was accomplished by using *fincore*, a tool that is part of the *linux-ftools* [8] suite, developed by Google. **RES** indicates the amount of memory occupied by the file contents; **PAGES** represents the amount of memory occupied by the file contents measured in pages; **SIZE** refers to the actual size of the file

Caching content in this way is typical, mainly due to its positive effects on I/O operations performance-wise. Working on content located in memory is objectively faster than working on content located on disk; this has been proven countless times (e.g., Apache Spark compared to Apache Hadoop [9]).

### 2.2.2 The Hot-Swap

Once the target file contents are in memory, the host will keep them there to perform any other operations on these contents for either: as long as the underlying operating system deems them necessary or if they are removed from memory due to other content having higher priority or requiring the space. Regardless of the scenario, the operating system decides when to do this based on its internal scheduler and scheduling algorithms. If we want to change the contents of *file.txt*, we need to remove it from memory on the host somehow. The obvious choice would be to remove the USB from the host, make the changes, and insert it back into the host. However, this would violate the principle of hot-swapping.

After changing the Pi’s target file, we need the host to remove its copy from memory. The way we have found to “encourage” the host operating system to remove the file contents from memory is to interact with other files that are *not* the target file, either on the same USB or on another USB. There are many ways to interact with USB contents: opening a file to read it, modifying the contents of a file, or opening a handle to the file for any reason. It is essential to get it in memory to convince the operating system to remove the contents of the target file stored in memory. The easiest method we have found is to calculate the hash

<sup>1</sup>In this case, cached implies that the contents have been stored on the host in a way that will improve the performance of operations performed on the contents. It does not refer to placing the contents in a hardware cache.

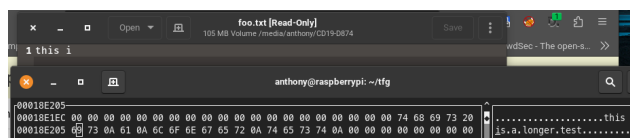
of files that are *not* the target file. This process can easily be implemented with a simple Bash and even PowerShell one-liner.

After some time (it can vary between a couple of seconds to a minute or two), the target file contents will be removed from memory, and when the user opens the target file on the host, the changes will be visible.

### 2.2.3 Limitations

As with any technique, there are some limitations regarding what modifications can be successfully performed. Namely:

- ➔ **File System Structure & Same Name:** Once the USB is inserted, one cannot change the structure of its contents. Consequently, *adding* to and *removing* files from the USB is impossible. Also, any changes to the target file cannot include its name. The name must remain the same, meaning we can only modify its contents. If these conditions are not respected, the hot swap will not work. We believe this is due to the metadata the file system has at the time the USB device is inserted into a computer. We have not found a way to hot-swap metadata, so we are subject to this limitation for the time being.
- ➔ **Modification Size:** The target file’s original size must be respected (and, obviously, the size of the USB file system) and not surpassed. For example, if the target file, upon creation, consisted of 10 bytes of content, we only have these 10 bytes of legroom. For argument’s sake, one can add more content to the target file, another 10 bytes, making the total size of content 20 bytes, but only the first 10 bytes of the new 20-byte file will be seen after the hot-swap is completed. We can see this in action below in figure 3.



**Fig. 3:** In this example, the file originally had 6 bytes of content “aaaaaa”. We can see here that the content was modified to “This is a longer test”, which is 21 bytes, and only the first 6 bytes of this new text was recognized by the host: “This i”.

## 3 FRAMEWORK DEVELOPMENT

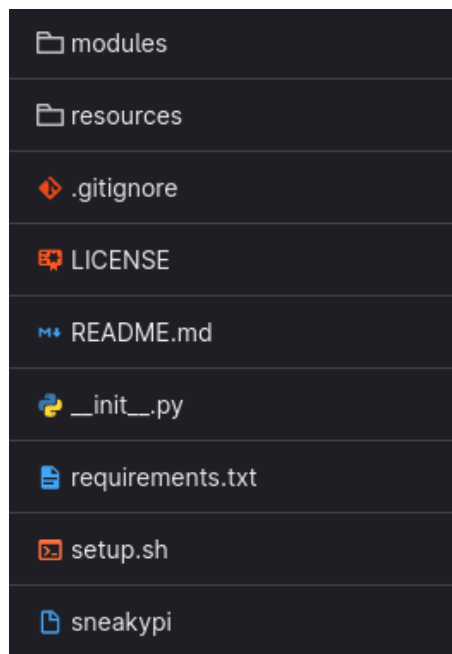
To help test this method’s limits, we wrote a framework named Sn34kyPi [10]. The framework design is heavily inspired by *msfconsole* [11], the console-based version of the Metasploit Framework. At the time of writing this document, Sn34kyPi comes with four main modules: *filesystem*, *creator*, *modifier*, and *usb*. It also includes scripts to facilitate setting up a few key components on the Pi: enabling and disabling a Wi-Fi hotspot, checking for the kernel modules required for the Pi to behave as a mass storage device, and setting them up if needed. The following subsections will explain the main components of the framework.

### 3.1 Why the Framework?

Performing the hot swap is not too complicated, especially for someone comfortable using a terminal. However, while it is relatively easy to accomplish, it can be challenging to manage. Imagine having a few different file systems to choose from, each of them of a different size and with a different number of files, and each of those with its content. On top of that, how does one create and manipulate files in a file system? The method we have grown accustomed to is to perform these actions on a mounted file system, implying the need for multiple directories to mount to or dismount the file system each time we want to mount another one. So now, we have multiple file systems with multiple files that need to be mounted to modify them. All this is far too complex to manage and efficiently test, which is where Sn34kyPi comes in. This framework is designed to take all the weight and problems off the shoulders of the user and provide an easy-to-use interface that provides the previously mentioned modules and scripts.

### 3.2 Framework Internals

Before we discuss the four main modules, we will provide a general overview of the framework's internals. First, we can see the directory structure in Figure 4.



**Fig. 4:** There are two main directories *modules* and *resources*. The former is where the framework's modules are stored, while the latter is where we can find notes, bash scripts, and Python resources.

The *modules* directory (which can easily be expanded by using the template module provided in *resources/python/template.py*) contains the four main modules of the framework, those of which will be reviewed in greater detail in the following subsections. The *resources* directory contains two sub-directories of interest: *bash\_scripts* and *python*. The Python files residing in *python* will be reviewed in section (3.2.6), while the scripts in *bash\_scripts* will be examined further in section (3.2.7). The four primary modules themselves do not have much code, given that *resources/python/super\_mod.py* provides

the baseline for new modules and is inherited by them, only requiring the author to override the methods: `execute()` and `short_description()`. The details of the requirements to create new modules have been defined in the *README.md* file located in the project root, under the section **Contributing**.

There are a few primary commands used to interact with modules: *load* `<module_name>` (used to load the specified module into the current context), *set* `<option> <value>` (to set an option of the currently loaded module to the specified value), *run* (used to execute the currently loaded module), *options* (to view the available options of the currently loaded module), and *unload* (unloads the currently loaded module from the current context).

#### 3.2.1 FileSystem

The *filesystem* module enables the user to create a FAT32 file system that will subsequently be used to contain any files the user wishes to create with the *creator* (3.2.2) module, modify with the *modifier* (3.2.3) module, and expose via an emulated USB with the *usb* (3.2.4) module. It first creates a container file using the *dd* tool, with the name and size (in MiB) specified by the user. Afterward, it converts this container file to a FAT32 file system. Figure 5 shows the options available to the user.

```
sneakypi (filesystem) > options
```

Property	Required	Current Value
fs_name	True	usb.bin
fs_size	True	500
fat_size	True	32

**Fig. 5:** There are three simple options available to the user: the file system name, the file system size (in MiB), and the FAT type. Comments have been cut off due to size restraints.

#### 3.2.2 Creator

This module expects the user to use a FAT-xx file system, as does the rest of the framework. Once run, it will open the specified file system and attempt to create the specified file of the user-defined size, referred to as “x” in this section. If the file already exists on the file system, it will notify the user to use the *modifier* module instead. If the file is created, it is filled with null bytes (`\x00`) until the file reaches the size requested by the user. We can see the available options in Figure 6.

When we originally wrote this module, it would fill the file with x “A”s. We then realized that when we change the file’s contents, if the new contents are less than the original file size, the remaining “A”s would be left there, causing unintended content to be left over post-modification that would either break the new content (if it is code) or mislead the user into thinking they inserted those “A”s themselves.



```
sneakypi (creator) > options
```

Property	Required	Current Value
filesystem	True	
file_name	True	
file_size	True	50

**Fig. 6:** Again, there are three options available to the user: the file system name, the name of the file they want to create, and the file size (in bytes). Comments have been cut off due to size restraints.

This realization caused the change to fill the file with null bytes instead.

### 3.2.3 Modifier

The user can modify a target file using the *modifier* module with command-line-provided content or a specified source file. In the first scenario, the provided content replaces the target file's contents, byte by byte. The latter first checks if the specified source file can be opened, and if so, it will copy the contents byte by byte to the target file. The options this module provides can be seen in Figure 7.

```
sneakypi (modifier) > options
```

Property	Required	Current Value
filesystem	True	
target	True	
content	False	
source	False	

**Fig. 7:** There are four options in this module: the file system name, the target file that the user wishes to modify, the content that will be placed into the target file (provided by the user via the command line), and a source file whose contents will be placed into the target file. The user can specify either the content or the source. Comments have been cut off due to size restraints.

### 3.2.4 USB

The *usb* module exposes the specified file system as a mass storage device, checking if it is FAT-xx beforehand, via the *g\_mass\_storage* kernel module. If this module is run and *g\_mass\_storage* has already been loaded, and the Pi is therefore already exposing a mass storage device, it will reload the kernel module, effectively removing and reinserting the USB device if the user chooses to do so. This module offers the options shown in Figure 8.

### 3.2.5 Global Commands

Global commands are those available to the user while using the framework, not necessarily in the context of a module. There are four of them, each described below:

```
sneakypi (usb) > options
```

Property	Required	Current Value
filesystem	True	
serial_number	False	1029384756
product_name	False	Sneaky USB
manufacturer_name	False	SneakyPi

**Fig. 8:** The *usb* module offers four options (three of them are not required): the file system name (required), the serial number, the product name, and the manufacturer name of the exposed mass storage device (not required). Comments have been cut off due to size restraints.

- ➔ **set fs:** The user can specify a FAT file system. Once this command is executed, Sn34kyPi will check if the provided value is a FAT file system before loading it into the global context. Once loaded into the global context, the rest of the global commands will interact with this file system, and any loaded modules will automatically use the specified file system by default.
- ➔ **show fs:** This command outputs the name of the currently globally loaded file system.
- ➔ **show files:** This command shows the files contained within the global file system in an easy-to-view list.
- ➔ **show stats:** This command outputs the stats of the global file system and its files. In Figure 9, we can see an example of what this command outputs.

```
sneakypi > show stats
```

Stats for linux_test.bin				
Filesystem Name	Filesystem Location	Number of files		
linux_test.bin	/home/anthony/Desktop/projects/tfg/linux_test.bin	2		
File Stats				
File name	File size	Creation Date	Modification Date	Metadata Changed
file1.sh	32B	2023-06-28 21:58:10	2023-06-28 21:58:10	
file2.bat	525B	2023-06-28 21:58:20	2023-06-28 21:58:20	

**Fig. 9:** The output of the *show stats* command, run after having loaded the example file system *linux\_test.bin*

### 3.2.6 python

The *python* directory contains four crucial Python files, which we can see in the following list:

- ➔ **super\_mod.py:** This is the parent class of all modules that are created in the *modules* directory. Its principal methods are *set()* and *get\_options()*. The former is invoked when the user runs the *set <option> <value>* command in the context of a loaded module, while the latter is called when the user runs the *options* command after having loaded a module into the current context. To gather the options of the module, we iterate over its attributes. It is important to note that it

is not sufficient to grab the attributes of the module itself but also the class attributes (which belong to the parent class, *SuperMod* in this case). Although *SuperMod* does not currently have its attributes, Python does distinguish them, so this is there in the case that during the framework's possible evolution, *SuperMod* does end up with properties.

- ➔ **properties.py**: This is a rather simple *dataclass* [13] I created to create properties (or attributes) for the modules. This *dataclass* defines a **name**, **value**, **comment**, and a **required** field. All fields are Strings except the **required** field, which is a Boolean. This design choice was taken to simplify the framework's creation of modules, avoiding creating a structure such as a Python dictionary to hold these values. This arrangement also helps to maintain consistency and avoid breakage if the user does not specify a comment or if the module is required, given that these two fields have default values. If the *options* command is run, the command will not break if the user has not put anything in these fields due to them having default values.
- ➔ **context.py**: Serves as the context for global commands. When the user sets the global file system, they interact with an instance of *Context*. This class was created instead of storing the global information in *SuperMod* since upon loading a module, *SuperMod* is instantiated along with the module (since the module inherits this class), effectively "erasing" the data we had stored. Using an instance of *Context*, we can easily pass it around, and the data will remain intact.
- ➔ **template.py**: As the name indicates, this class serves as a template for creating new modules. There is little to explain here.

### 3.2.7 bash\_scripts

There are four primary bash scripts included in Sn34kyPi:

- ➔ **hotspot.sh**: This script enables a Wi-Fi hotspot on the Pi. It is important to remember that Pi Zero W cannot simultaneously provide Internet access and a Wi-Fi hotspot. This is because this Pi does not have an Ethernet port; therefore, its network card must be used either for itself or dedicated to the Wi-Fi hotspot. This side effect can be quite bothersome at times since the user cannot access the Internet while connected to the Pi's hotspot, but currently, there is no way to avoid this side effect.

An essential aspect of getting the hotspot to work is the power source of the Pi. After exhaustive testing, we concluded that some USB ports do not provide enough voltage or amperage to the Pi, leading to strange behavior such as the hotspot turning off or not even turning on. From what we could gather, most computers' USB ports are *powered* [12], while some ports are "*unpowered*". These "*unpowered*" ports provide a meager amount of power, but not enough for the Pi to function reliably, and we believe the computers we saw this issue on have "*unpowered*" USB ports.

- ➔ **hotspot-removal.sh** The name is self-explanatory; it turns off the services required for the hotspot to be enabled (*hostapd* and *dnsmasq*) and comments out the lines written to */etc/dhccpd.conf* for the hotspot to function, effectively turning off the hotspot and allowing the Pi to connect to some other Wi-Fi network.
- ➔ **mass\_storage\_requirements.sh** This must be run to enable the Pi to act as a mass storage device when requested. Mainly, it makes sure the kernel modules required for **g\_mass\_storage** to run are loaded at boot time, namely **i2c-dev** and **dwc2**. If these kernel modules are not loaded, trying to run **g\_mass\_storage** will produce an error similar to: "*udc-core: some error message*".
- ➔ **setup.sh** This script checks for two things: ensuring the Pi is running Bullseye (which this framework has been successfully tested on) and that the Pi has an Internet connection. It then proceeds to install the necessary packages to create a virtual environment for the user, after which it will activate the Python virtual environment and download the necessary Python packages the framework relies on to function correctly. This is all done in the current working directory of the user. We chose to work in a virtual Python environment to not "contaminate" any of the underlying system's Python packages and to avoid any version conflict between them.

## 4 SN34KYPI DEMONSTRATION

In this section, we will show off Sn34kyPi with a demonstration. In this demo, the bash scripts do not need to be executed because we have already set up the Pi using them. We have also already connected the Pi to a computer (so it is running), and it is not emitting a Wi-Fi hotspot.

First, we will connect to the Pi via SSH and start the framework. Note that we start the framework as *sudo* because we require root privileges to interact with kernel modules (i.e., **g\_mass\_storage**). We can visualize this first step in Figure 10.

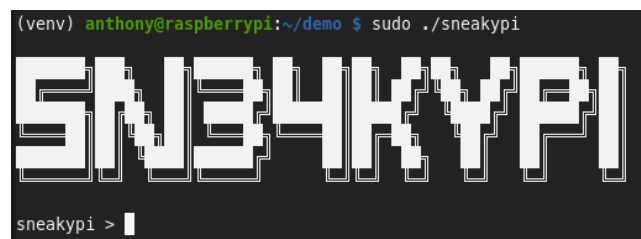


Fig. 10: Starting the framework on the Pi.

After starting the framework, let us create a FAT32 file system, as shown in Figure 11.

We will create some files in **demo.bin**, as displayed in Figure 12.

There is a reason the file extension for the second file is **.bat**, and we will see its implications shortly. Let us continue the demo by modifying our new files as shown in Figure 13 and Figure 14.

We can verify the contents of, for example, **file2.bat**, as shown in Figure 15.

```
sneakypi > load filesystem
sneakypi (filesystem) > set fs_size 1000
fs_size ==> 1000
sneakypi (filesystem) > set fs_name demo.bin
fs_name ==> demo.bin
sneakypi (filesystem) > run
Creating 1000MiB container file demo.bin...
[✓] Successfully created container file demo.bin
Converting demo.bin to a DOS filesystem...
[✓] Successfully converted demo.bin to a DOS filesystem
Global filesystem set to: demo.bin
sneakypi (filesystem) > □
```

**Fig. 11:** Loading the *filesystem* module and executing it to create a FAT32 filesystem of 1000 MiB named **demo.bin**

```
sneakypi (filesystem) > load creator
sneakypi (creator) > set file_name file1.txt
file_name ==> file1.txt
sneakypi (creator) > run
[✓] Created file1.txt of 50 bytes in demo.bin!
sneakypi (creator) > set file_name file2.bat
file_name ==> file2.bat
sneakypi (creator) > set file_size 600
file_size ==> 600
sneakypi (creator) > run
[✓] Created file2.bat of 600 bytes in demo.bin!
sneakypi (creator) > □
```

**Fig. 12:** Creating two files: **file1.txt** and **file2.bat**, of 50 bytes and 600 bytes respectively.

```
sneakypi (creator) > load modifier
sneakypi (modifier) > set target file1.txt
target ==> file1.txt
sneakypi (modifier) > set source /home/anthony/ps.ps1
source ==> /home/anthony/ps.ps1
sneakypi (modifier) > run
Opened source file /home/anthony/ps.ps1...
Copying from /home/anthony/ps.ps1 to file1.txt...
[✓] Successfully copied content to file1.txt!
sneakypi (modifier) > □
```

**Fig. 13:** With the *modifier* module, we copy the contents from the file **/home/anthony/ps.ps1** to **file1.txt**

```
sneakypi (modifier) > set target file2.bat
target ==> file2.bat
sneakypi (modifier) > set source /home/anthony/rev.bat
source ==> /home/anthony/rev.bat
sneakypi (modifier) > run
Opened source file /home/anthony/rev.bat...
Copying from /home/anthony/rev.bat to file2.bat...
[✓] Successfully copied content to file2.bat!
sneakypi (modifier) > □
```

**Fig. 14:** With the *modifier* module, we copy the contents from the file **/home/anthony/rev.bat** to **file2.bat**

Now, we can load the USB and interact with its files on our computer (a laptop with a Linux distribution installed). We will do this with the *usb* module, as seen in Figure 16. We can see that everything is working as expected: the device detected is 1 GiB in size, and it has the two files we created and modified in the previous steps.

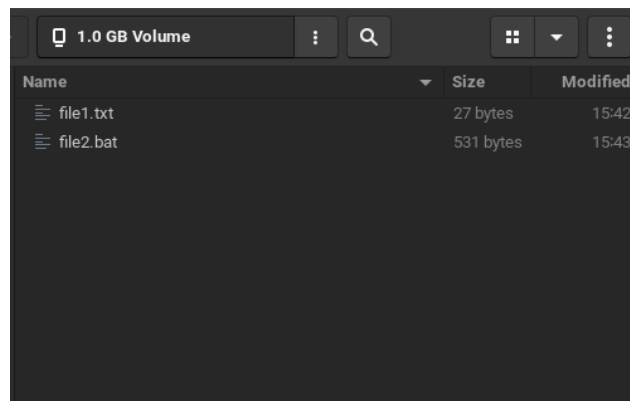
We can see something interesting if we view the USB contents in our terminal (on the host computer, not the Pi), as shown in Figure 17. Notice that the executable bit is enabled on **file2.bat** for all users, meaning we can execute the file from the terminal, while **file1.txt** does not have the executable bit set! This is not expected behavior since the FAT

```
sneakypi (modifier) > get file2.bat
file2.bat contents:
powershell -nop -c "$client = New-Ob
$client.GetStream();[byte[]]$bytes =
-ne 0){;$data = (New-Object -Type Name
(iex $data 2>&1 | Out-String );$sendb
(::ASCII).GetBytes($sendback2);$stre
e())"
pause
```

**Fig. 15:** Verifying the contents of **file2.bat** within the framework.

standard does not support POSIX permissions [14] [15]. This is exciting! (More on how we discovered this in section 4.1.)

Let us take advantage of this and **hot-swap** the contents of the executable file (**file2.bat**) with a simple PoC bash script using the *modifier* module. We will set the content of **file2.bat** to our PoC, which will print the string “Hello, friend” (Mr.Robot reference).



```
sneakypi (modifier) > load usb
sneakypi (usb) > run
Checking if demo.bin is a valid filesystem...
[✓] demo.bin looks good...
Checking if g_mass_storage is already loaded...
Exposing demo.bin...
[✓] Exposed USB! You should see the device on the host.
sneakypi (usb) > □
```

**Fig. 16:** Viewing the exposed USB device, performed with the *usb* module.

```
> cd /media/anthony/E72C-5B62
> ls -la
total 16
drwxr-xr-x  2 anthony anthony 4096 Jan  1  1970 .
drwxr-xr-x  3 root    root    4096 Jun 30 14:08 ..
-rw-r--r--  1 anthony anthony 27 Jun 30 2023 file1.txt
-rwxr-xr-x  1 anthony anthony 531 Jun 30 2023 file2.bat
```

**Fig. 17:** Viewing the exposed USB contents in the host terminal. We can see that **file2.bat** has the executable bit enabled for all users.

Figure 18 shows how the framework easily enables us to replace a target file’s contents. After executing the module, we need to remove the copy of **file2.bat** that the host operating system stored in the host’s memory, as discussed in section 2.2.2.



```

Name                               Size    Modified
file1.txt                          27 bytes  15:42
file2.bat                          531 bytes  15:43

[✓] Exposed USB! You should see the device on the host.
sneakypi (usb) > load modifier
sneakypi (modifier) > set target file2.bat
target ==> file2.bat
sneakypi (modifier) > set content "#!/bin/bash\nHello, friend"
content ==> #!/bin/bash
echo Hello, friend
sneakypi (modifier) > run
Modifying content of file2.bat...
[✓] Successfully modified content of file2.bat!

```

**Fig. 18:** Using the *modifier* module to insert our PoC bash script into **file2.bat**

According to the plan, we will insert another USB device into the host computer and run a quick bash one-liner (command used can be seen in listing (1) in the new USB directory to calculate the hash of its files. We can see this in action in Figure 19.

**Listing 1:** Bash one-liner to calculate the SHA256 hash of all files in the current directory

```
find . -type f -exec sha256sum {} \;
```

```

sneakypi (modifier) > set target file2.bat
target ==> file2.bat
sneakypi (modifier) > set content "#!/bin/bash\nHello, friend"
content ==> #!/bin/bash
echo Hello, friend
sneakypi (modifier) > run
Modifying content of file2.bat...
[✓] Successfully modified content of file2.bat!
sneakypi (modifier) >

> sudo sysctl vm.drop_caches=3
vm.drop_caches = 3
> find . -type f -exec sha256sum {} \;
3bb420e02babf9398f43752135057de0dec59a72aaca79b41964b6d1ef060ba0 ./.
9e4c1d2040ab5ec4e92068f611348cf9f8f72b79e7f84156737f507d04323df5 ./.
9e4c1d2040ab5ec4e92068f611348cf9f8f72b79e7f84156737f507d04323df5 ./.
^C

```

**Fig. 19:** Calculating the SHA256 hash of all files in the other USB device. We stopped it once we saw that the contents of **file2.bat** were removed from the host memory.

After some time (we can see precisely when the contents are removed from the host memory using the *fincore* tool we saw in Figure 2), the new contents will appear on the host, after which we can execute the PoC content we inserted before in Figure 20.

## 4.1 Notes

After multiple failed attempts to acquire code execution through batch files on Windows (an example we thought would be engaging and exciting), we decided that for the demo's sake, we would create a simple bash PoC instead

```

[✓] Exposed USB! You should see the device on the host.
sneakypi (usb) > load modifier
sneakypi (modifier) > set target file2.bat
target ==> file2.bat
sneakypi (modifier) > set content "#!/bin/bash\nHello, friend"
content ==> #!/bin/bash
echo Hello, friend
sneakypi (modifier) > run
Modifying content of file2.bat...
[✓] Successfully modified content of file2.bat!
sneakypi (modifier) >

> ./file2.bat
Hello, friend

```

**Fig. 20:** Voilà!

(which is what we showed in this section). After a few days, we opened the USB device in the terminal and noticed that the batch files we created had the executable bit set! We determined that this occurs in the following scenario: After making a simple batch file, it is read-only if we view it in Linux. If we open the file in a Windows machine and double-click on it from File Explorer, even if it does not correctly execute, the executable bit is set when we view the file on the Linux machine again. This requires more investigation, which is why this framework was created!

## 5 CONCLUSIONS

After our extensive research, it is clear that hot-swapping a USB file system opens the door to various possible vulnerabilities. With the intent of discovering these vulnerabilities, the framework created as a result of this project will help us, and hopefully, any other researchers, uncover them. This framework indeed requires some touching up, but in its current state, it can pave the way for further investigation into USB file system hot-swapping. When researching cybersecurity-related topics, as in our case, it is important to try everything thoroughly, pay attention to the details, as insignificant as they may seem, and not let the possibility of failure deter one from investigating. If we had not considered this subject, we would have missed what could be quite an exciting discovery.

## ACKNOWLEDGEMENTS

I thank my tutor Carles Martinez Garcia for introducing me to the original hot-swapping technique, leading to the start of this project. Before this project, I had not contemplated the possibility of something so seemingly ordinary being vulnerable, but this experience has proved that everything is “hackable”.

In addition to Carles, I would like to thank Porfidio Hernández Budé for his time in helping me understand what was occurring under the hood while performing the hot-swapping. Although we could not fully determine what transpires behind the scenes, his insights were invaluable for me to understand further what may be happening, and discussing the topic with another knowledgeable person aided me further.

Finally, I thank my family and friends for enduring my endless, sometimes incoherent, jabbering about all topics related to this project.

## REFERENCES

- [1] BitDefender, “Bitdefender Security Software Solutions for Home Users,” Bitdefender. <https://www.bitdefender.com/solutions/>
- [2] NIST, “Security Considerations for Code Signing,” 2018. Available: <https://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/01/26/security-considerations-for-code-signing/final/documents/security-considerations-for-code-signing.pdf>
- [3] The Kernel Development Community, “Mass Storage Gadget (MSG) — The Linux Kernel documentation,” [www.kernel.org](http://www.kernel.org). <https://www.kernel.org/doc/html/latest/usb/mass-storage.html> (accessed Jun. 19, 2023).
- [4] The Raspberry Pi Foundation, “Buy a Raspberry Pi Zero W – Raspberry Pi,” [Raspberrypi.org](http://Raspberrypi.org), 2017. <https://www.raspberrypi.org/products/raspberry-pi-zero-w/>
- [5] Kubii, “Adaptador ZeroKey USB para Pi Zero,” KUBII. <https://www.kubii.com/es/hub-cables-adaptadores/2063-adaptador-zerokey-usb-para-pi-zero-3272496009271.html> (accessed Jun. 19, 2023).
- [6] Raspberry Pi Ltd, “Operating system images,” Raspberry Pi. <https://www.raspberrypi.com/software/operating-systems/>
- [7] Raspberry Pi Ltd, “Raspberry Pi OS,” Raspberry Pi. <https://www.raspberrypi.com/software/>
- [8] Google, “Google Code Archive - Long-term storage for Google Code Project Hosting.,” [code.google.com](http://code.google.com), 2010. <https://code.google.com/archive/p/linux-ftools/> (accessed Jun. 23, 2023).
- [9] IBM Cloud Education, “Hadoop vs. Spark: What’s the Difference?,” [www.ibm.com](http://www.ibm.com), May 27, 2021. <https://www.ibm.com/cloud/blog/hadoop-vs-spark>
- [10] AMAlonso64, “Anthony Alonso / sneakypi · GitLab,” GitLab, 2023. <https://gitlab.com/AMAlonso64/sneakypi>
- [11] “rapid7/metasploit-framework,” GitHub, Aug. 14, 2020. <https://github.com/rapid7/metasploit-framework>
- [12] “USB,” Wikipedia, Mar. 19, 2022. <https://en.wikipedia.org/wiki/USB>
- [13] “dataclasses — Data Classes — Python 3.8.3 documentation,” [docs.python.org](http://docs.python.org). <https://docs.python.org/3/library/dataclasses.html>
- [14] Deland-Han, veganaize, AmandaAZ, and simonxjx, “Overview of FAT, HPFS, and NTFS File Systems - Windows Client,” [learn.microsoft.com](http://learn.microsoft.com), Sep. 23, 2021. <https://learn.microsoft.com/en-us/troubleshoot/windows-client/backup-and-storage/fat-hpfs-and-ntfs-file-systemsfat-overview>
- [15] “File-system permissions,” Wikipedia, Jun. 24, 2023. [https://en.wikipedia.org/wiki/File-system\\_permissions#File\\_system\\_variations](https://en.wikipedia.org/wiki/File-system_permissions#File_system_variations)