out-of-tree

unknown

Feb 28, 2024

CONTENTS

out-of-tree is the kernel {module, exploit} development tool.

out-of-tree was created on the purpose of decreasing complexity of environment for developing, testing and debugging Linux kernel exploits and out-of-tree kernel modules (that's why tool got a name "out-of-tree").

While I'm trying to keep that documentation up-to-date, there may be some missing information. Use `out-of-tree --help-long` for checking all features.

If you found anything missed here, please make a pull request or send patches to patch@dumpstack.io.

If you need personal support, your company is interested in the project or you just want to share some thoughts – feel free to write to root@dumpstack.io.

CHAPTER

ONE

CONTENTS

Keyword Index

1.1 Introduction

out-of-tree is written in *Go*, it uses *Docker* for generating kernel/filesystem images and *Qemu* for virtualization.

Also it possible to generate kernels from the host system and use the custom one.

out-of-tree supports *GNU/Linux* (usually it's tested on NixOS and latest Ubuntu LTS) and *macOS*. Technically all systems that supported by Go, Docker, and Qemu must work well. Create the issue if you'll notice any issue in integration for your operating system.

All *Qemu* interaction is stateless.

out-of-tree is allow and require metadata (`.out-of-tree.toml`) for work. TOML (Tom's Obvious, Minimal Language) is used for kernel module/exploit description.

`.out-of-tree.toml` is mandatory, you need to have in the current directory (usually, it's a project of kernel module/exploit) or use the `--path` flag.

1.1.1 Files

All data is stored in `~/.out-of-tree/`.

- *db.sqlite* contains logs related to run with `out-of-tree pew`, debug mode (`out-of-tree debug`) is not store any data.
- *images* used for filesystem images (rootfs images that used for `qemu -hda ...`) that can be generated with the `tools/qemu-*-img/...`.
- *kernels* stores all kernel `vmlinuz/initrd/config/...` files that generated previously with a some *Docker magic*.
- *kernels.toml* contains metadata for generated kernels. It's not supposed to be edited by hands.
- *kernels.user.toml* is default path for custom kernels definition.
- *Ubuntu* (or *Centos*/*Debian*/…) is the Dockerfiles tree (DistroName/DistroVersion/Dockerfile). Each Dockerfile contains a base layer and incrementally updated list of kernels that must be installed.

1.1.2 Overview

out-of-tree creating debugging environment based on **defined** kernels:

```
$ out-of-tree debug --kernel 'Ubuntu:4.15.0-58-generic'
[*] KASLR SMEP SMAP
[*] gdb is listening on tcp::1234
[*] build result copied to /tmp/exploit

ssh -o StrictHostKeyChecking=no -p 29308 root@127.133.45.236
gdb /usr/lib/debug/boot/vmlinux-4.15.0-58-generic -ex 'target remote tcp::1234'

out-of-tree> help
help    : print this help message
log     : print qemu log
clog    : print qemu log and cleanup buffer
cleanup : cleanup qemu log buffer
ssh     : print arguments to ssh command
quit    : quit
out-of-tree>
```

out-of-tree uses three stages for automated runs:

- Build
  - Inside the docker container (default).
  - Binary version (de facto skip stage).
  - On host.
- Run
  - Insmod for the kernel module.
  - This step is skipped for exploits.
- Test
  - Run the test.sh script on the target machine.
  - Test script is run from *root* for the kernel module.
  - Test script is run from *user* for the kernel exploit.
  - Test script for the kernel module is fully custom (only return value is checked).
  - Test script for the kernel exploit receives two parameters:
    - Path to exploit
    - Path to file that must be created with root privileges.
  - If there's no test.sh script then default (`echo touch FILE | exploit`) one is used.

1.1.3 Security

out-of-tree is not supposed to be used on multi-user systems or with an untrusted input.

Meanwhile, all modern hypervisors are supporting nested virtualization, which means you can use it for isolating *out-of-tree* if you want to work with an untrusted input (e.g. with a mass-scale testing public proofs-of-concept).

1.2 Installation (from source)

1.3 OS/Distro-specific

1.3.1 Ubuntu

Install dependencies:

```
$ sudo snap install go --classic
$ # Install docker: https://docs.docker.com/engine/install/ubuntu/
$ sudo apt install qemu-system-x86 build-essential gdb
```

1.3.2 macOS

Install dependencies:

```
$ brew install go qemu
$ brew cask install docker
```

1.3.3 NixOS

There's a minimal configuration that you need to apply:

```
#!nix
{ config, pkgs, ... }:
{
  virtualisation.docker.enable = true;
  virtualisation.libvirtd.enable = true;
  environment.systemPackages = with pkgs; [
    go git
  ];
}
```

1.3.4 Gentoo

Install dependencies:

```
$ sudo emerge app-emulation/qemu app-emulation/docker dev-lang/go
```

1.3.5 Fedora

Install dependencies:

```
$ sudo dnf install go qemu moby-engine
```

1.4 Common

Setup environment:

```
$ echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
$ source ~/.bashrc
```

Build *out-of-tree*:

```
$ git clone https://code.dumpstack.io/tools/out-of-tree
$ cd out-of-tree
$ CGO_ENABLED=1 go build -o ~/bin/out-of-tree
```

Note: On a GNU/Linux you need to add your user to docker group if you want to use *out-of-tree* without sudo. Note that this has a **serious** security implications. Check *Docker* documentation for more information.

Test that everything works:

```
$ cd out-of-tree/examples/kernel-exploit
$ out-of-tree kernel autogen --max=1
$ out-of-tree pew --max=1
```

Enjoy!