DRAKVUF Sandbox

Release v0.16.1

CERT Polska

May 11, 2021

USER GUIDE

DRAKVUF Sandbox is an automated black-box malware analysis system with DRAKVUF engine under the hood, which does not require an agent on guest OS.

This project provides you with a friendly web interface that allows you to upload suspicious files to be analyzed. Once the sandboxing job is finished, you can explore the analysis result through the mentioned interface and get insight whether the file is truly malicious or not.

Because it is usually pretty hard to set up a malware sandbox, this project also provides you with an installer app that would guide you through the necessary steps and configure your system using settings that are recommended for beginners. At the same time, experienced users can tweak some settings or even replace some infrastructure parts to better suit their needs.

CHAPTER

ONE

GETTING STARTED

1.1 Supported hardware & software

In order to run DRAKVUF Sandbox, your setup must fullfill all of the listed requirements:

- Processor: Intel processor with VT-x and EPT features (*how to check*).
- Host system: Debian 10 Buster/Ubuntu 18.04 Bionic/Ubuntu 20.04 Focal with at least 2 core CPU and 5 GB RAM, running GRUB as bootloader.
- Guest system: Windows 7 (x64), Windows 10 (x64; experimental support)

Nested virtualization:

- KVM **does** work, however it is considered experimental. If you experience any bugs, please report them to us for further investigation.
- Due to lack of exposed CPU features, hosting DRAKVUF Sandbox in the cloud is **not** supported (although it might change in the future).
- Hyper-V does **not** work.
- Xen **does** work out of the box.
- VMware Workstation Player **does** work, but you need to check Virtualize EPT option for a VM; Intel processor with EPT still required.

1.2 Basic installation

This instruction assumes that you want to create a single-node installation with the default components, which is recommended for beginners.

1. Download latest release packages.
2. Install DRAKVUF:

```
# apt update
# apt install ./drakvuf-bundle*.deb
# reboot
```

3. Install DRAKVUF Sandbox stack:

```
# apt install redis-server
# apt install ./drakcore*.deb
# apt install ./drakrun*.deb
```

4. Check if your Xen installation is compliant. This command should print "All tests passed":

```
# draksetup test
```

5. Execute:

```
# draksetup install /opt/path_to_windows.iso
```

Read the command's output carefully. This command will run a virtual machine with Windows system installation process.

Customize vCPUs/memory: You can pass additional options in order to customize number of vCPUs (`--vcpus <number>`) and amount of memory (`--memory <num_mbytes>`) per single VM. For instance: `--vcpus 1 --memory 2048`.

Recommended minimal values that are known to work properly with DRAKVUF Sandbox:

System version	Minimal vCPUs	Minimal RAM
Windows 7	1	1536
Windows 10	2	3072

Unattended installation: If you have `autounattend.xml` matching your Windows ISO, you can request unattended installation by adding `--unattended-xml /path/to/autounattend.xml`. Unattended install configuration can be generated with Windows Answer File Generator.

Note: By default, DRAKVUF Sandbox will store virtual machine's HDD in a `qcow2` file. If you want to use ZFS instead, please check the *ZFS storage backend* docs.

---

6. Use VNC to connect to the installation process:

```
$ vncviewer localhost:5900
```

7. Perform Windows installation until you are booted to the desktop.
8. **Optional:** At this point you might optionally install additional software. You can execute:

```
# draksetup mount /path/to/some-cd.iso
```

which would mount a virtual CD disk containing additional software into your VM.

9. **Optional:** Generate .NET Framework native image cache by executing the following commands in the administrative prompt of your VM.

```
cd C:\Windows\Microsoft.NET\Framework\v4.0.30319
ngen.exe executeQueuedItems
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319
ngen.exe executeQueuedItems
```

10. In order to finalize the VM setup process, execute:

```
# draksetup postinstall
```

Note: Add `--no-report` if you don't want `draksetup` to send basic usage report.

---

11. Test your installation by navigating to the web interface (http://localhost:6300/) and uploading some samples. The default analysis time is 10 minutes.

CHAPTER

TWO

OPTIONAL FEATURES

This sections contains various information about optional features that may be enabled when setting up DRAKVUF Sandbox.

2.1 ZFS storage backend

If you want to install DRAKVUF Sandbox with a ZFS storage backend, you should perform the following extra steps before executing `draksetup install` command:

1. Install ZFS on your machine (guide for: Debian Buster, Ubuntu 18.04)
2. Create a ZFS pool on a free partition:

   ```
   # zpool create tank <partition_name>
   ```

   where `<partiton_name>` is e.g. `/dev/sda3`. Be aware that all data stored on the selected partition may be erased.
3. Create a dataset for DRAKVUF Sandbox:

   ```
   # zfs create tank/vms
   ```

4. Execute `draksetup install` as in "Basic installation" section, but remembering to provide additional command line switches:

   ```
   --storage-backend zfs --zfs-tank-name tank/vms
   ```

2.2 Networking

Note: Even though that the guest Internet connectivity is an optional feature, `drakrun` would always make some changes to your host system's network configuration:

Always:

- Each instance of `drakrun@<vm_id>` will create a bridge `drak<vm_id>`, assign `10.13.<vm_id>.1/24` IP address/subnet to it and bring the interface up.
- `drakrun` will drop any INPUT traffic originating from `drak<vm_id>` bridge, except DHCP traffic (UDP ports: 67, 68).

Only with `net_enable=1`:

- `drakrun` will enable IPv4 forwarding.
- `drakrun` will configure MASQUERADE through `out_interface` for packets originating from `10.13.<vm_id>.0/24`.
- `drakrun` will DROP traffic between `drak<X>` and `drak<Y>` bridges for `X != Y`.

In order to find out the exact details of the network configuration, search for `_add_iptable_rule` function usages in `drakrun/drakrun/main.py` file.

2.2.1 Basic networking

If you want your guest VMs to access Internet, you can enable networking by editing `[drakrun]` section in `/etc/drakrun/config.ini`:

- Set `net_enable=1` in order to enable guest Internet access.
- Check if `out_interface` was detected properly (e.g. `ens33`) and if not, correct this setting.

After making changes to `/etc/drakrun`, you need to restart all `drakrun` services that are running in your system:

```
# systemctl restart 'drakrun@*'
```

Be aware that if your sandbox instance is already running some analyses, the above command will gracefully wait up to a few minutes until these are completed.

2.2.2 Using dnschef

You may optionally configure your guests to use dnschef.

1. Setup dnschef tool.
2. Start `dnschef` in such way to make it listen on all `drak*` interfaces that belong to DRAKVUF Sandbox.
3. Set `dns_server=use-gateway-address` in `/etc/drakrun/config.ini`.
4. Restart your drakrun instances: `systemctl restart 'drakrun@*`.

2.3 MS Office file support

There is an experimental support for analyzing word and excel samples. However this requires that you have Microsoft Office installed.

The steps below should be completed on guest vm before creating the snapshot (e.g. before you run `draksetup postinstall`).

1. Install Microsoft Office. You can use `draksetup mount /path/to/office.iso` command to insert Office installation media during VM setup. After installation, you should be able to start word/excel by running `start winword.exe`, `start excel.exe` from command line.
2. Adjust the registry keys by executing this *.reg* file:

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Word\Security]
"VBAWarnings"=dword:00000001
"AccessVBOM"=dword:00000001
"ExtensionHardening"=dword:00000000

[HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Excel\Security]
"VBAWarnings"=dword:00000001
"AccessVBOM"=dword:00000001
"ExtensionHardening"=dword:00000000
```

(change 14.0 to your Office version, see registry key by product name)

2.4 ProcDOT integration

DRAKVUF Sandbox may optionally draw a behavioral graph using ProcDOT, if `drakcore` will find it's binary installed at `/opt/procdot/procmon2dot`.

1. Download ProcDOT (Linux version).
2. With your downloaded `procdot*_linux.zip` archive, execute the following commands:

```
# unzip -o procdot*_linux.zip lin64/* -d /tmp/procdot
# mv /tmp/procdot/lin64 /opt/procdot
# chmod +x /opt/procdot/procmon2dot
```

3. Your new analysis reports will also contain behavioral graphs.

CHAPTER

THREE

MANAGING SNAPSHOTS

3.1 Snapshot modification

Before trying to modify the installation, make sure that all `drakrun@` services are stopped.

Execute `drakplayground 0` as root. Output of the command should look similarly to this:

```
dnsmasq: started, version 2.83 DNS disabled
dnsmasq: compile time options: IPv6 GNU-getopt DBus no-UBus i18n IDN2 DHCP DHCPv6 no-
↪Lua TFTP conntrack ipset auth nettlehash DNSSEC loop-detect inotify dumpfile
dnsmasq-dhcp: DHCP, IP range 10.13.0.100 -- 10.13.0.200, lease time 12h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface drak0
Loading new save file /var/lib/drakrun/volumes/snapshot.sav (new xl fmt info 0x3/0x0/
↪2015)
 Savefile contains xl domain config in JSON format
Parsing config from /etc/drakrun/configs/vm-0.cfg
xc: info: Found x86 HVM domain from Xen 4.15
xc: info: Restoring domain
xc: info: Restore successful
xc: info: XenStore: mfn 0xfeffc, dom 0, evt 1
xc: info: Console: mfn 0xfefff, dom 0, evt 2
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.9.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

You will be dropped into a IPython shell, with vm-0 running and internet connection configured. At this point you can connect to VNC and perform the modifications. **Don't exit** the shell or close the terminal.

If you have some scripts, executables or other files on the host, you can copy them into the VM with a helper function:

```
In [1]: copy("/root/examples/example1.exe")
```

Copied files should appear on the desktop.

When you're done, open another terminal window and execute `draksetup postinstall`. The command will recreate the snapshot and profiles for other virtual machines.

It is now safe to close the shell. To do this execute:

```
In [10]: exit()
```

or hit Ctrl+D.

Warning: vm-0 is a base for other virtual machines. Leaving it in a broken or inconsistent state will result in analysis failures, BSODs and other unexpected errors. When modifying the vm-0 always make sure to perform the postinstall step.

3.2 Importing and exporting snapshots

Current sandbox implemention allows for a single VM snapshot installed on a machine. However, it is possible to export and import snapshots from a remote server.

This is especially useful when running drakrun on multiple machines that should share same snapshot.

There are two types of snapshots: minimal and full. Before doing anything you should know which is appropriate for your usecase.

3.2.1 Minimal snapshot

Minimal snapshot contains only the most essential parts of the virtual machine which include HDD image and VM configuration.

This has both some advantages and drawbacks:

- before using the snapshot on a new machine, VM must be cold booted to the desktop and `draksetup postinstall` must be executed to extract runtime information,
- this snapshot type is more portable and stable as the operating system is being booted on the hardware that will be used for performing analyses.

Note: Starting minimal VM may trigger operating system checks for a dirty filesystem. This shouldn't cause any issues after configuring the snapshot.

3.2.2 Full snapshot

Full snapshots contain all of the data required by `drakrun` to work correctly. Apart from configuration and disk images they also contain compressed dumps of the VM's physical memory and runtime information.

After importing a full snapshot no additional steps are required.

Warning: Full snapshots are tightly coupled with the hardware they were generated on. Importing incompatible snapshot may result in unexpected behavior ranging from failures to create virtual machies, to guest crashes.

When in doubt use minimal snapshots

CHAPTER

FOUR

SCALING

4.1 Introduction

After performing installation, by default, your sandbox instance will be capable of processing one sample at a time. The service that performs the actual analysis is called *drakrun@<instance_number>*. You can check the state of a particular instance by executing:

```
systemctl status drakrun@1
```

You can change the number of parallel workers by executing:

```
draksetup scale <num_instances>
```

4.2 Scaling up

Assuming you have a single instance but you want to be able to process 10 samples in parallel, you should execute:

```
draksetup scale 10
```

The setup script will configure and start additional instances named from `drakrun@2` to `drakrun@10`.

4.3 Scaling down

Analogously, you can scale down by repeating the same command with the smaller number of instances, e.g.:

```
draksetup scale 7
```

Assuming you had 10 instances previously, it will cause `drakrun@8` to `drakrun@10` to be disabled and shut down. If the analysis is pending on these instances, the command will gracefully wait until it's finished.

4.4 Postprocess

Analysis postprocessing doesn't need hypervisor access, so it can be done in separate servers, assuming they have same configuration and connect to same minio & redis instances. This is highly recommended if you can afford such setup, as this frees resources on servers running hypervisor.

By default only 1 instance of postprocess worker is started and when running multiple instances of drakrun - needs to be scaled up. As a rule of thumb you can assume safe ratio of postprocess to drakrun workers to be 1:3 (however, this ratio can vary depending on performance of the platform and analysis duration). To startup more postprocessing instances just start more instances of `drak-postprocess@` service. By default only 1 is present, so be sure to scale it accordingly to your needs.

The following command will start second postprocessing worker.

```
systemctl enable --now drak-postprocess@2
```

CHAPTER

FIVE

TROUBLESHOOTING

5.1 Checking service status

If your DRAKVUF Sandbox installation seems to work improperly, here are some commands that would help to troubleshoot the infrastructure.

Check service status:

```
# drak-healthcheck
```

Check service logs:

```
# journalctl -e -u drak-web
# journalctl -e -u drak-system
# journalctl -e -u drak-minio
# journalctl -e -u drak-postprocess@1
# journalctl -e -u drakrun@1
```

5.2 Debug `device model did not start`

You may encounter the following error with `draksetup` command or `drakrun@*` service, which will prevent the VM from starting properly.

```
libxl: error: libxl_create.c:1676:domcreate_devmodel_started: Domain 4:device model␣
↪did not start: -3
...
subprocess.CalledProcessError: Command 'xl create /etc/drakrun/configs/vm-0.cfg'␣
↪returned non-zero exit status 3.
```

In such a case, you should inspect `/var/log/xen/qemu*.log` in order to determine the actual reason why the VM is refusing to start.

5.3 Debug `can't allocate low memory for domain`

The following error with `draksetup` command or `drakrun@*` service means that your machine is missing memory resources:

```
xc: error: panic: xc_dom_boot.c:122: xc_dom_boot_mem_init: can't allocate low memory␣
↪for domain: Out of memory
...
subprocess.CalledProcessError: Command 'xl create /etc/drakrun/configs/vm-0.cfg'␣
↪returned non-zero exit status 3.
```

Resolutions:

- adjust the amount of memory dedicated to the Dom0 (host system) in `/etc/default/grub.d/xen.cfg` (look for `dom0_mem=2048M,max:2048M`) and run `update-grub && reboot`
- adjust the amount of memory dedicated to the DomU (guest systems) in `/etc/drakrun/scripts/cfg.template` (`maxmem` and `memory` keys)

CHAPTER

SIX

UPGRADING

We strive to make the installation and upgrade process as simple as possible, so in order to use a new version you have to perform just a few steps.

> **Warning:** Always install correct package versions and perform all upgrade steps. Mismatching packages from different releases may lead to unexpected results.

Before upgrading the sandbox, stop the sandbox workers:

```
# systemctl stop drakrun@*
```

> **Note:** If some analyses are running, the command will block until they've finished.

Install new packages and reboot:

```
# apt install ./drakvuf-bundle*.deb
# apt install ./drakrun*.deb
# apt install ./drakcore*.deb
# systemctl reboot
```

After rebooting, make sure that all of the services are running with a command:

```
# drak-healthcheck
```

CHAPTER

SEVEN

UNDERSTANDING THE SANDBOX

7.1 Tech stack

DRAKVUF Sandbox is built on top of a few layers of software and hardware technologies:

- Intel VT-x and EPT - extensions to x64 architecture that allow to run virtual machines natively on a CPU
- Xen - hypervisor, spawns virtual machines and exposes interfaces for interaction and introspection
- LibVMI - abstracts away introspection interfaces, provides utilities for reading/writing VM memory, parsing VMs' kernel and handling notifications about certain events happening in a VM
- DRAKVUF - stealthily hooks various parts of a guest VM and logs interesting events
- DRAKVUF Sandbox - provides user friendly interface and high level analyses

7.2 Project structure

DRAKVUF Sandbox is divided into two packages:

- drakcore - system core, provides a web interface, an internal task queue and object storage
- drakrun - sandbox worker, wrapper for DRAKVUF, responsible for managing VMs, running analyses and sending results for further postprocessing.

Note: DRAKVUF engine is a separate project authored by Tamas K Lengyel.

DRAKVUF Sandbox is built around karton – microservice framework created at CERT Poland as a specialized tool for building flexible malware analysis pipelines. Its main goal is routing tasks between multiple services.

7.3 Daemons

- drakcore package
  - `drak-web` - web interface that allows user to interact with the sandbox with either REST API or GUI
  - `drak-system` - internal task management system, using for dispatching jobs between workers
  - `drak-minio` - builtin object storage in which analysis results are stored
  - `drak-postprocess` - responsible for processing raw analysis logs into more usable form

- drakrun package
  - `drakrun 1..n` - fetches incoming samples for analysis, runs VMs, and sends back results of analysis; each daemon handles one concurrent VM

7.4 Lifecycle of a analysis

1. User submits new analysis with a browser or programatically using *karton* API.
2. `drak-system` dispatches the job to one of the `drakrun` instances.
3. `drakrun` runs the analysis:
   - preconfigured virtual machine image is restored
   - sample is uploaded to the VM using DRAKVUF's `injector`
   - sample is executed
   - after a chosen timeout, virtual machine is destroyed
4. Raw results (dumps, logs, pcaps) are sent back to `drak-system` as a *karton* task.
5. `drak-system` dispatches a task to `drak-postprocess` which extracts interesting data for the user

CHAPTER

EIGHT

SANDBOX DEVELOPMENT

DRAKVUF Sandbox is not a typical monolithic application. It is designed to be deployed over multiple servers either standalone or as a part of a larger karton system. Multiple components and daemons may be confusing at the beginning.

This is a quick tutorial that should help you when starting to develop the sandbox.

DRAKVUF Sandbox is based on karton framework. It is recommended to become familiar with its concepts before approaching the sandbox code.

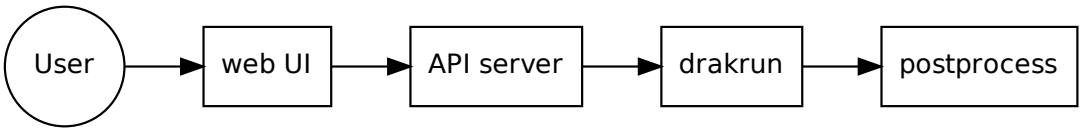

Fig. 1: High-level view of component interactions

8.1 Web UI (drakcore)

Serves as an GUI for the user for sample submission and browsing the results. Built with React and Hyper bootstrap theme.

Code location: drakcore/drakcore/frontend

8.1.1 Development

The prerequisite is to setup a working DRAKVUF Sandbox instance (MinIO, Redis, drakrun and API). Workflow is going to be similar to developing other React-based apps with a backend API. Don't forget to run Prettier over the changes. Otherwise CI will reject your code.

```
$ cd drakcore/drakcore/frontend
$ # install dependencies (execute only the first time)
$ npm install
$ # point the application at a running instance of API server
$ export REACT_APP_API_SERVER=http://[API location]:6300/
$ # start serving the frontend with live reloading
$ npm start
```

8.2 REST API (drakcore)

Main entrypoint into the sandbox. The intended users are web UI and programmatic integrations with the sandbox.

Code location: drakcore/drakcore/app.py

8.2.1 Development

REST API is a simple Flask-based Python application. To work correctly it requires a configuration file (stored in `/etc/drakcore/config.ini` on a configured sandbox instance) to reach to Karton and drakrun workers. If you want to run the API server on a different machine than it is originally configured you may have to tweak it a little.

```
$ # Create python virtualenv
$ python -m venv venv
$ source env/bin/activate
$ cd drakcore
$ # Copy the configuration file to the same directory as config.dist.ini
$ cp /some/config.ini drakcore/config.ini
$ # Install drakcore dependencies
$ pip install -r requirements.txt
$ # Install drakcore in editable mode
$ pip install -e .
$ export FLASK_APP=drakcore/app.py
$ export FLASK_ENV=development
$ flask run
```

8.3 drakrun (drakrun)

This is the main component that manages the analysis process and the only one that has the requirement of being deployed on a machine (either virtual or physical) running Xen.

Code location: drakrun/drakrun

8.3.1 Development

This is the hardest part to develop as it has to be on a running on a separate machine. First, setup the basic environment in the repository:

```
$ # Make sure that installed drakrun instance is not running
$ systemctl stop drakrun@1
$ # Create Python virtualenv
$ python -m venv venv
$ source env/bin/activate
$ cd drakrun
$ # Install drakrun dependencies
$ pip install -r requirements.txt
$ # Install drakrun in editable mode
$ pip install -e .
$ # Start drakrun
$ python drakrun/main.py 1
```

drakrun should start listening for new task from the rest of the system. After making some changes you have to restart the process.

To develop drakrun from your main development machine you can either:

- mount the repository directory over SSHFS
- use an IDE integration to edit remote files
- (advanced) add the drakrun repository on a worker machine as another Git remote and push the changes

8.4 postprocessing (drakcore)

Hypervisor time is precious. This is why it's important to perform as little work as possible in drakrun process. Analysis postprocessing extracts interesting data from DRAKVUF output and converts it into a form that is easier to consume by the frontend.

8.4.1 Development

On a drakrun machine:

```
$ # Make sure that the installed drak-postprocess instance is not running
$ systemctl stop drak-postprocess@1
```

On a development machine:

```
$ # Create python virtualenv
$ python -m venv venv
$ source env/bin/activate
$ cd drakcore
$ # Copy the configuration file to the same directory as config.dist.ini
$ cp /some/config.ini drakcore/config.ini
$ # Install drakcore dependencies
$ pip install -r requirements.txt
$ # Install drakcore in editable mode
$ pip install -e .
$ # Start the postprocess worker
$ python process.py
```

Code location (entrypoint): drakcore/drakcore/process.py

Code location (steps): drakcore/drakcore/postprocess

CHAPTER

NINE

REGRESSION TESTING

9.1 Introduction

Memory dumping is one of the core functionalities used for automated malware analysis. Unpacked or decrypted memory is saved for futher analysis with YARA rules or configuration extraction. Thus, it's important to ensure that DRAKVUF development does cause any regressions that would break existing sample analysis.

9.2 Preparing a test set

Regression test set is a list of JSON objects that represent a number of sample submissions and the expected malware family name that should be detected.

Dump analysis is performed by providing a directory with malduck extractor modules. Here, you can learn more about them.

- sha256 - SHA256 hash of the sample file
- extension - file extension, supported by the sanbox, e.g. "exe" or "dll"
- ripped - malware family name
- path - (optional) path to the malware sample

Example:

```
[
    {
        "sha256": "35e756ef1b3d542deaf59f093bc4abe5282a1294f7144b32b61f4f60c147cabb",
        "extension": "dll",
        "ripped": "emotet"
    },
    {
        "sha256": "4239335443cbf3d45db485d33c13346c67d5ac717a57856315a166c190dde075",
        "extension": "exe",
        "ripped": "raccoon",
        "path": "samples/
↪4239335443cbf3d45db485d33c13346c67d5ac717a57856315a166c190dde075"
    }
]
```

Test submitter supports two methods for obtaining the malware sample.

1. Manual - if the test case has a `path` key deinfed, malware sample will be read from this location (relative and absolute paths are allowed).

2. Automated - otherwise, sample will be downloaded from the mwdb.cert.pl service. Make sure to run the submitter with `MWDB_API_KEY` environment variable if you intend to use this method

9.3 Running the receiver daemon

First, configure the extractor module path in `/etc/drakrun/config.ini`

```
[draktestd]
; path to the extraction modules for
; https://github.com/CERT-Polska/malduck
modules=/opt/extractor-modules/
```

Next, uncomment `sample_testing` line and enable it

```
[drakrun]
; (advanced) Enable testing codepaths. Test sample artifacts will not be uploaded
; to persistent storage. Their lifetime will be bound to karton tasks produced by␣
↪drakrun
sample_testing=1
```

Then, execute

```
$ draktestd
```

This will spawn a new karton service listening for test analysis results and printing the results.

9.4 Executing a test set

To submit a test set, execute:

```
$ draktest test_set.json
```

The command will submit samples to the sandbox and wait until all the testing is finished.

CHAPTER

TEN

BUILDING INSTALLATION PACKAGES

In order to build installation packages on your own, you must first install Docker on your machine.

10.1 DRAKVUF Sandbox (drakcore, drakrun)

You may build your packages from source using following commands:

```
$ git clone https://github.com/CERT-Polska/drakvuf-sandbox.git
$ cd drakvuf-sandbox
$ sudo ./drakcore/package/build.sh
$ sudo ./drakrun/package/build.sh
```

Afterwards, you should find your installation packages produced in *out/* directory.

10.2 DRAKVUF (drakvuf-bundle)

The build scripts for *drakvuf-bundle* are part of tklengyel/drakvuf repository. You may build your package using the following commands:

```
$ git clone --recursive https://github.com/tklengyel/drakvuf
$ cd drakvuf
$ sudo ./package/build.sh
```

The resulting package will be produced to `package/out/` directory.

CHAPTER

ELEVEN

DRAKVUF SANDBOX FAQ

11.1 Can I run DRAKVUF Sandbox in the cloud?

We've done some research regarding the deployment of the sandbox in the cloud. Unfortunately, due to the nature of the project and extensive use of low level CPU features, none of the popular "instance" services were able to run DRAKVUF. If you're interested to learn more about underlying problems see relevant issues on GitHub.

However, this doesn't mean that cloud deployment is impossible. You can still leverage modern deployment techniques and IaC (infrastracture as code) using bare metal servers.

Tested service providers:

- Equinix Metal
- Scaleway Bare Metal

Unfortunately, AWS EC2 Metal seems to be broken at the moment (see this issue). If you've managed to run DRAKVUF Sandbox on a previously untested cloud service, send us a PR to add it to this list.

11.2 How can I verify if my CPU is supported?

If you're running fairly recent Intel CPU, it's probably going to have all of the required features.

0. Make sure VT-x extensions are enabled in BIOS.
1. Check virtualization extensions support.

```
$ lscpu | grep vmx
```

2. Check EPT support.

```
$ lscpu | grep ept
```

If both flags are present, you're good to go.

11.3 I have an AMD CPU which supports NPT. Can I run DRAKVUF Sandbox?

DRAKVUF is tightly coupled with alpt2m feature, implemented only for Intel CPUs. Thus it's not possible to run it on a AMD CPU.

11.4 I have some other question

Feel free to submit an issue, write us an email or contact in any other way.

CHAPTER

TWELVE

USING DRAKPDB TOOL

The `drakpdb` tool allows you to:

- determine PDB name and GUID age given an executable file (e.g. DLL)
- fetch PDB with given name and GUID age
- parse PDB into a profile that could be plugged into DRAKVUF

12.1 Usage examples

```
root@zen2:~/drakvuf# drakpdb pdb_guid --file ntdll.dll
{'filename': 'wntdll.pdb', 'GUID': 'dccff2d483fa4dee81dc04552c73bb5e2'}
root@zen2:~/drakvuf# drakpdb fetch_pdb --pdb_name wntdll.pdb --guid_age␣
↪dccff2d483fa4dee81dc04552c73bb5e2
100%|████████████████████████████████████████████████████████████| 2.12M/2.12M␣
↪[00:00<00:00, 2.27MiB/s]
root@zen2:~/drakvuf# drakpdb parse_pdb --pdb_name wntdll.pdb > profile.json
```

CHAPTER

THIRTEEN

USING INTEL PROCESSOR TRACE FEATURES (EXPERIMENTAL)

13.1 Enable IPT plugin in drakrun

1. In `/etc/drakrun/config.ini`, add `ipt` plugin under `[drakvuf_plugins]` section `__all__` in order to enable IPT tracing.
2. In `/etc/drakrun/scripts/cfg.template` add a new entry: `vmtrace_buf_kb = 8192`
3. Execute `systemctl restart drakrun@1` (repeat for each drakrun instance if you have scaled them up).

13.2 Install required extra dependencies

In order to analyze IPT data streams, you need to install `libipt`, `xed`, `ptdump` (modified), `ptxed` and `drak-ipt-blocks` tools.

```
rm -rf /tmp/iptbuild
mkdir /tmp/iptbuild
cd /tmp/iptbuild

git clone https://github.com/icedevml/libipt.git
git clone https://github.com/intelxed/xed.git
git clone https://github.com/intelxed/mbuild.git
git clone https://github.com/gabime/spdlog.git
git clone https://github.com/CERT-Polska/drakvuf-sandbox.git

cd xed
./mfile.py --share
./mfile.py --prefix=/usr/local install
ldconfig

cd ../libipt
git checkout
cmake -D PTDUMP=On -D PTXED=On .
make install

cd ../spdlog
cmake .
make -j$(nproc) install

cd ../drakvuf-sandbox/drakcore/drakcore/tools/ipt
cmake .
make install
```

13.3 Generate trace disassembly

1. Perform an analysis with IPT plugin enabled
2. Download the completed analysis from MinIO to your local hard drive
3. Find CR3 of the target process you want to disassemble (hint: *syscall.log* will contain CR3 values)
4. Execute `drak-ipt-disasm --analysis . --cr3 <target_process_cr3> --vcpu 0`
5. After few minutes it should start printing full trace disassembly of the targeted process
6. You can also try *–blocks* switch for *drak-ipt-disasm* to get a list of executed basic blocks for this process

Example (executed basic blocks):

```
# drak-ipt-disasm --analysis . --cr3 0x735bb000 --vcpu 0 --blocks
[2021-04-19 23:47:41.717] [console] [info] Decoding
{ "event": "block_executed", "data": "0x7feff565088" }
{ "event": "block_executed", "data": "0x7feff75450f" }
{ "event": "block_executed", "data": "0x7feff754505" }
{ "event": "block_executed", "data": "0x7feff75450d" }
{ "event": "block_executed", "data": "0x7feff5656ac" }
{ "event": "block_executed", "data": "0x7feff5656dc" }
{ "event": "block_executed", "data": "0x7feff5656fb" }
{ "event": "block_executed", "data": "0x7feff565068" }
{ "event": "block_executed", "data": "0x7feff751530" }
{ "event": "block_executed", "data": "0x7feff751552" }
...
```

Example (full usermode disassembly):

```
# drak-ipt-disasm --analysis . --cr3 0x735bb000 --vcpu 0 | grep -v ptwrite | grep -v␣
↪cbr
[enabled]
[exec mode: 64-bit]
000007feff565088  movdqu xmmword ptr [rip+0x1b2b80], xmm0
000007feff565090  ret
000007feff75450f  add rbx, 0x8
000007feff754513  cmp rbx, rdi
000007feff754516  jb 0x7feff754505
000007feff754505  mov rax, qword ptr [rbx]
000007feff754508  test rax, rax
000007feff75450b  jz 0x7feff75450f
...
```