Slips

Release 1.0.13

Stratosphere Laboratory

Apr 28, 2024

SLIPS

The tool is available on GitHub here.

Slips is a Python-based intrusion prevention system that uses machine learning to detect malicious behaviors in the network traffic. Slips was designed to focus on targeted attacks, to detect of command and control channelsi, and to provide good visualisation for the analyst. Slips is able to analyze real live traffic from the device and the large network captures in the type of a pcap files, Suricata, Zeek/Bro and Argus flows. As a result, Slips highlights suspicious behaviour and connections that needs to be deeper analyzed.

This documentation gives an overview how Slips works, how to use it and how to help. To be specific, that table of contents goes as follows:

- **Installation**. Instructions to install Slips in a Docker and in a computer. See *Installation*.
- **Usage**. Instructions and examples how to run Slips with different type of files and analyze the traffic using Slips and its GUI Kalipso. See *Usage*.
- **Detection modules**. Explanation of detection modules in Slips, types of input and output. See *Detection modules*.
- **Architecture**. Internal architecture of Slips (profiles, timewindows), the use of Zeek and connection to Redis. See *Architecture*.
- **Training with your own data**. Explanation on how to re-train the machine learning system of Slips with your own traffic (normal or malicious).See *Training*.
- **Detections per Flow**. Explanation on how Slips works to make detections on each flow with different techniques. See *Flow Alerts*.
- **Exporting**. The exporting module allows Slips to export to Slack and STIX servers. See *Exporting*.
- **Slips in Action**. Example of using slips to analyze different PCAPs See *Slips in action*.
- **Contributing**. Explanation how to contribute to Slips, and instructions how to implement new detection module in Slips. See *Contributing*.
- **Create a new module**. Step by step guide on how to create a new Slips module See *Create a new module*.
- **Code documentation**. Auto generated slips code documentation See *Code docs*.

CHAPTER

ONE

INSTALLATION

There are two ways to install and run Slips: inside a Docker or in your own computer. We suggest to install and to run Slips inside a Docker since all dependencies are already installed in there. However, current version of docker with Slips does not allow to capture the traffic from the computer's interface. We will describe both ways of installation anyway.

1.1 Table of Contents

1.2 Requirements

Slips requires Python 3.8+ and at least 4 GBs of RAM to run smoothly.

1.3 Slips in Docker

Slips can be run inside a Docker. Either using our docker image with from DockerHub (recommended) or building Slips image from the Dockerfile for more advanced users.

In both cases, you need to have the Docker platform installed in your computer. Instructions how to install Docker is https://docs.docker.com/get-docker/.

The recommended way of using slips would be to

- *Run Slips from Dockerhub*

For more advanced users, you can:

- *Run Slips using docker compose*
- *Build Slips using the dockerfile*

1.3.1 Running Slips from DockerHub

1. First, choose the correct image for your architecture

For linux

Analyse your own traffic

```
- `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-swap="8g
↪" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output -v
↪$(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/slips:latest␣
↪/StratosphereLinuxIPS/slips.py -i eno1`
- Please change the name of the interface for your own.
- Check the alerts slips generated
  - ```tail -f output/eno1*/alerts.log ```
```

Analyze your PCAP file

```
- Prepare a dataset directory
        - `mkdir dataset`
        - `cp myfile.pcap dataset`
  - Run Slips
        - `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-
↪swap="8g" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output␣
↪-v $(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/
↪slips:latest /StratosphereLinuxIPS/slips.py -f dataset/myfile.pcap`
  - Check the alerts slips generated
          - ```tail -f output/myfile*/alerts.log ```
```

For MacOS M1

Analyse your own traffic

```
- `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-swap="8g
↪" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output -v
↪$(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/slips:latest␣
↪/StratosphereLinuxIPS/slips.py -i eno1`
- Please change the name of the interface for your own.
- Check the alerts slips generated
  - ```tail -f output/eno1*/alerts.log ```

docker run -it --rm --net=host stratosphereips/slips_macos_m1:latest
```

Docker with P2P is not supported for MacOS M1.

For MacOS Intel processors

Analyse your own traffic

```
- `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-swap="8g
↪" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output -v
↪$(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/slips:latest␣
↪/StratosphereLinuxIPS/slips.py -i eno1`
- Please change the name of the interface for your own.
- Check the alerts slips generated
  - ```tail -f output/eno1*/alerts.log ```
```

Analyze your PCAP file

```
- Prepare a dataset directory
        - `mkdir dataset`
        - `cp myfile.pcap dataset`
  - Run Slips
        - `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-
↪swap="8g" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output␣
↪-v $(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/
↪slips:latest /StratosphereLinuxIPS/slips.py -f dataset/myfile.pcap`
  - Check the alerts slips generated
          - ```tail -f output/myfile*/alerts.log ```
```

For P2P support on Linux

To analyze your own traffic with p2p

~~~
- `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-swap="8g
↪" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output -v
↪$(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/slips_
↪p2p:latest /StratosphereLinuxIPS/slips.py -i eno1 -o output_dir `
- Please change the name of the interface for your own.
- Check evidence
  ```tail -f output_dir/alerts.log ```
~~~

For P2P support on MacOS Intel

Analyze your own traffic

~~~
- `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-swap="8g
↪" --net=host --cap-add=NET_ADMIN -v $(pwd)/output:/StratosphereLinuxIPS/output -v
↪$(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/slips_
↪p2p:latest /StratosphereLinuxIPS/slips.py -i eno1 -o output_dir `
- Please change the name of the interface for your own.
- Check evidence
  ```tail -f output_dir/alerts.log ```
~~~

Once your image is ready, you can run slips using the following command:

```
./slips.py -f dataset/dataset/test7-malicious.pcap
```

To analyze your own file using slips, you can mount it to your docker using -v

```
mkdir ~/dataset
cp <some-place>/myfile.pcap ~/dataset
docker run -it --rm --net=host -v ~/dataset:/StratosphereLinuxIPS/dataset␣
↪stratosphereips/slips:latest
./slips.py -f dataset/myfile.pcap
```

1.3.2 Updating the image in case there is a new one

```
docker pull stratosphereips/slips:latest
```

1.3.3 Known Error in old GPUs

If you happen to get the error `Illegal instruction (core dumped)` it means that tensorflow can not be run from inside Docker in your GPU. We recommend to disable the modules using machine learning by modifying the `disable` line in the configuration to be like this `disable = [template, ensembling, rnn-cc-detection, flowmldetection]`

If you were running slips directly from the docker without cloning the repo, you can do this modification in two ways:

1. Modify the container
   1. Run the docker in background using the same command as above but with `-d`
   2. Get into the docker with `docker exec -it slips /bin/bash`, and then modifying the configuration file in `config/slips.conf` to add the disabled modules
   3. Run Slips from inside the docker `./slips.py -i enp7s0`
2. You can
   1. Clone the Slips repo (clone the same version as the docker you are downloading),
   2. Modify your local `config/slips.conf`
   3. Run the docker command above but by mounting the volume of the config. `docker run --rm -it -p 55000:55000 --cpu-shares "700" --memory="8g" --memory-swap="8g" --net=host --cap-add=NET_ADMIN -v $(pwd)/config:/StratosphereLinuxIPS/config/ -v $(pwd)/output:/StratosphereLinuxIPS/output -v $(pwd)/dataset:/StratosphereLinuxIPS/dataset --name slips stratosphereips/slips:latest /StratosphereLinuxIPS/slips.py -i eno1`

1.3.4 Run Slips sharing files between the host and the container

The following instructions will guide you on how to run a Slips docker container with file sharing between the host and the container.

```
    # create a directory to load pcaps in your host computer
    mkdir ~/dataset

    # copy the pcap to analyze to the newly created folder
    cp <some-place>/myfile.pcap ~/dataset

    # create a new Slips container mapping the folder in the host to a folder in the␣
↪container
    docker run -it --rm --net=host --name slips -v $(pwd)/dataset:/StratosphereLinuxIPS/
↪dataset stratosphereips/slips:latest

    # run Slips on the pcap file mapped to the container
    ./slips.py -f dataset/myfile.pcap
```

1.3.5 Run Slips with access to block traffic on the host network

In Linux OS, the Slips can be used to analyze and **block** network traffic on the host network interface. To allow the container to see the host interface traffic and block malicious connections, it needs to run with the option `--cap-add=NET_ADMIN`. This option enables the container to interact with the network stack of the host computer. To block malicious behavior, run Slips with the parameter `-p`.

Change eno1 in the command below to your own interface

```
  # run a new Slips container with the option to interact with the network stack of␣
↪the host
  docker run -it --rm --net=host --cap-add=NET_ADMIN --name slips stratosphereips/
↪slips:latest

  # run Slips on the host interface `eno1` with active blocking `-p`
  ./slips.py -i eno1 -p
```

1.3.6 Running Slips using docker compose

Change enp1s0 to your current interface in docker/docker-compose.yml and start slips using

```
docker compose -f docker/docker-compose.yml up
```

Now everything inside your host's `config` and `dataset` directories is mounted to `/StratosphereLinuxIPS/config/` and `/StratosphereLinuxIPS/dataset/` in Slips docker.

To run slips on a pcap instead of your interface you can do the following:

1. put the pcap in the `dataset/` dir in your host
2. change the entrypoint in the docker compose file to [”python3”,”/StratosphereLinuxIPS/slips.py”,”-f”,”dataset/.pcap”]
3. restart slips using `docker compose -f docker/docker-compose.yml up`

Limitations

The main limitation of running Slips in a Docker is that every time the container stops, all files inside the container are deleted, including the Redis database of cached data, and you lose all your Threat Intelligence (TI) data and previous detections. Next time you run Slips, it will start making detections without all the TI data until downloading the data again. The only solution is to keep the container up between scans.

1.3.7 Building Slips from the Dockerfile

First, you need to check which image is suitable for your architecture.

Before building the docker locally from the Dockerfile, first you should clone Slips repo or download the code directly:

```
git clone https://github.com/stratosphereips/StratosphereLinuxIPS.git
```

If you cloned Slips in '~/code/StratosphereLinuxIPS', then you can build the Docker image with:

NOTE: replace ubuntu-image with the image that fits your archiecture

```
cd ~/code/StratosphereLinuxIPS/docker/ubunutu-image
docker build --no-cache -t slips -f Dockerfile .
docker run -it --rm --net=host -v ~/code/StratosphereLinuxIPS/dataset:/
↪StratosphereLinuxIPS/dataset slips
./slips.py -c config/slips.conf -f dataset/test3-mixed.binetflow
```

If you don't have Internet connection from inside your Docker image while building, you may have another set of networks defined in your Docker. For that try:

```
docker build --network=host --no-cache -t slips -f Dockerfile .
```

You can also put your own files in the /dataset/ folder and analyze them with Slips:

```
cp some-pcap-file.pcap ~/code/StratosphereLinuxIPS/dataset
docker run -it --rm --net=host -v ../dataset/:/StratosphereLinuxIPS/dataset slips
./slips.py -f dataset/some-pcap-file.pcap
```

Note that some GPUs don't support tensorflow in docker which may cause "Illegal instruction" errors when running slips.

To fix this you can disable all machine learning based modules when running Slips in docker, or run Slips locally.

1.4 Installing Slips natively

Slips is dependent on three major elements:

Python 3.8 Zeek Redis database 7.0.4

To install these elements we will use APT package manager. After that, we will install python packages required for Slips to run and its modules to work. Also, Slips' interface Kalipso depend on Node.JS and several npm packages.

Instructions to download everything for Slips are below.

1.4.1 Install Slips using shell script

You can install it using install.sh

```
sudo chmod +x install.sh
sudo ./install.sh
```

1.4.2 Installing Slips manually

Installing Python, Redis, NodeJs, and required python and npm libraries.

Update the repository of packages so you see the latest versions:

```
apt-get update
```

Install the required packages (-y to install without asking for approval):

```
apt-get -y install tshark iproute2 python3.8 python3-tzlocal net-tools python3-dev build-
↪essential python3-certifi curl git gnupg ca-certificates redis wget python3-minimal␣
↪python3-redis python3-pip python3-watchdog nodejs redis-server npm lsof file iptables␣
↪nfdump zeek whois yara
apt install -y --no-install-recommends nodejs
```

Even though we just installed pip3, the package installer for Python (3.8), we need to upgrade it to its latest version:

```
python3 -m pip install --upgrade pip
```

Now that pip3 is upgraded, we can proceed to install all required packages via pip3 python packet manager:

```
sudo pip3 install -r install/requirements.txt
```

Note: for those using a different base image, you need to also install tensorflow==2.2.0 via pip3.

As we mentioned before, the GUI of Slips known as Kalipso relies on NodeJs v19. Make sure to use NodeJs greater than version 12. For Kalipso to work, we will install the following npm packages:

```
curl -fsSL https://deb.nodesource.com/setup_21.x |  sudo -E bash - && sudo apt install -
↪y --no-install-recommends nodejs
cd modules/kalipso &&  npm install
```

Installing Zeek

The last requirement to run Slips is Zeek. Zeek is not directly available on Ubuntu or Debian. To install it, we will first add the repository source to our apt package manager source list. The following two commands are for Ubuntu, check the repositories for the correct version if you are using a different OS:

```
echo 'deb http://download.opensuse.org/repositories/security:/zeek/xUbuntu_18.04/ /' |␣
↪tee /etc/apt/sources.list.d/security:zeek.list
```

We will download and store the gpg signature from the package for apt to read:

```
curl -fsSL http://download.opensuse.org/repositories/security:/zeek/xUbuntu_18.04/
↪Release.key | gpg --dearmor | tee /etc/apt/trusted.gpg.d/security_zeek.gpg > /dev/null
```

Finally, we will update the package manager repositories and install zeek

```
apt-get update
apt-get -y install zeek
```

To make sure that zeek can be found in the system we will add its link to a known path:

```
ln -s /opt/zeek/bin/zeek /usr/local/bin
```

Running Slips for the First Time

Be aware that the first time you run Slips it will start updating all the databases and threat intelligence files in the background. However, it will give you as many detections as possible *while* updating. You may have more detections if you rerun Slips after the updates. Slips behaves like this, so you don't have to wait for the updates to finish to have some detections. however, you can change that in the config file by setting `wait_for_TI_to_finish` to yes.

Depending on the remote sites, downloading and updating the DB may take up to 4 minutes. Slips stores this information in a cache Redis database, which is kept in memory when Slips stops. Next time Slips runs, it will read from this database. The information in the DB is updated periodically according to the configuration file (usually one day).

You can check if the DB is running this by looking at your processes:

```
    ps afx | grep redis
    9078 ?        Ssl    1:25 redis-server *:6379
```

You can kill this redis database by running:

```
    ./slips.py -k
    Choose which one to kill [0,1,2 etc..]
    [0] Close all servers
    [1] conn.log - port 6379
```

then choosing 1.

1.5 Installing Slips on a Raspberry PI

Slips on RPI is currently in beta and is actively under development. While it is functional, please be aware that there may be occasional bugs or changes in functionality as we work to improve and refine this feature. Your feedback and contributions are highly valuable during this stage!

Instead of compiling zeek, you can grab the zeek binaries for your OS

Packages for Raspbian 11:

https://download.opensuse.org/repositories/security:/zeek/Raspbian_11/armhf/zeek_4.2.1-0_armhf.deb

Packages for Raspbian 10:

https://download.opensuse.org/repositories/security:/zeek/Raspbian_10/armhf/zeek_4.2.1-0_armhf.deb

CHAPTER

TWO

USAGE

Slips can read the packets directly from the **network interface** of the host machine, and packets and flows from different types of files, including

- Pcap files (internally using Zeek)
- Packets directly from an interface (internally using Zeek)
- Suricata flows (from JSON files created by Suricata, such as eve.json)
- Argus flows (CSV file separated by commas or TABs)
- Zeek/Bro flows from a Zeek folder with log files
- Nfdump flows from a binary nfdump file
- Text flows from stdin in zeek, argus or suricata form

It's recommended to use PCAPs.

All the input flows are converted to an internal format. So once read, Slips works the same with all of them.

After Slips was run on the traffic, the Slips output can be analyzed with Kalipso GUI interface. In this section, we will explain how to execute each type of file in Slips, and the output can be analyzed with Kalipso.

Either you are running Slips in docker or locally, you can run Slips using the same below commands and configurations.

2.1 Reading the input

The table below shows the commands Slips uses for different inputs. The first part of the command **./slips.py -c config/slips.conf** is same, the second part changes depending on the input type. Also, the user can execute **./slips.py –help** to find correct argument to run Slips on each type of the file.

(*) To find the interface in Linux, you can use the command `ifconfig`.

There is also a configuration file **config/slips.conf** where the user can set up parameters for Slips execution and models separately. Configuration of the **config/slips.conf** is described *here*.

2.2 Daemonized vs interactive mode

Slips has 2 modes, interactive and daemonized.

Daemonized : means , output, logs and alerts are written in files.

In daemonized mode : Slips runs completely in the background, The output is written to `stdout`, `stderr` and `logsfile` files specified in `config/slips.conf`

by default, these are the paths used

stdout = /var/log/slips/slips.log stderr = /var/log/slips/error.log logsfile = /var/log/slips/slips.log

NOTE: Since `/val/log/` is owned by root by default, If you want to store the logs in `/var/log/slips`, creat /var/log/slips as root and slips will use it by default.

If slips can't write there, slips will store the logs in the `Slips/output/` dir by default.

NOTE: if -o <output_dir> is given when slips is in daemonized mode, the output log files will be stored in <output_dir> instead of the otput_dir specified in config/slips.conf

This is the not the default mode, to use it, run slips with -D

```
./slips.py -i wlp3s0 -D
```

To stop the daemon run slips with `-S`, for example `./slips.py -S`

Only one instance of the daemon can be running at a time.

Interactive : For viewing output, logs and alerts in a terminal, usually used for developers and debugging.

This is the default mode, It doesn't require any flags.

Output files are stored in `output/` dir.

By default you don't need root to run slips, but if you changed the default output directory to a dir that is owned by root, you will need to run Slips using sudo or give the current user enough permission so that slips can write to those files.

For detailed information on how slips uses redis check the Running several slips instances section

2.3 Running Several slips instances

By default, Slips will assume you are running only 1 instance and will use the redis port 6379 on each run.

You can run several instances of slips at the same time using the -m flag, and the output of each instance will be stored in `output/filename_timestamp/` directory.

If you want Slips to run on a certain port, you can use the `-P <portnumber>` parameter to specify the port you want Slips to use. but it will always use port 6379 db 1 for the cache db.

Each instance of Slips will connect to redis server on a randomly generated port in the range (32768 to 32850).

In macos, you will get a popup asking for permission to open and use that random port, press yes to allow it.

However, all instance share 1 cached redis database on redis://localhost:6379 DB 1, to store the IoCs taken from TI files.

Both redis servers, the main sever (DB 0) and the cache server (DB 1) are opened automatically by Slips.

When running ./kalipso.sh, you will be prompted with the following

```
To close all unused redis servers, run slips with --killall
You have 3 open redis servers, Choose which one to use [1,2,3 etc..]
[1] wlp3s0 - port 55879
[2] dataset/test7-malicious.pcap - port 59324
```

You can type 1 or 2 to view the corresponding file or interface in kalipso.

Once you're done, you can run slips with `--killall` to close all the redis servers using the following command

`./slips.py --killall`

NOTICE: if you run more than one instance of Slips on the same file or the same interface, Slips will generate a new directory with the name of the file and the new timestamp inside the `output/` dir

2.4 Closing redis servers

Slips uses a random unused port in the range in the range (32768 to 32850).

When running slips, it will warn you if you have more than 1 redis serve open using the following msg

```
[Main] Warning: You have 2 redis servers running. Run Slips with --killall to stop them.
```

you can use the -k flag to kill 1 open redis server, or all of them using the following command

```
./slips.py -k
```

You will be prompted with the following options

```
Choose which one to kill [0,1,2 etc..]

[0] Close all servers
[1] dataset/sample_zeek_files - port 32768
[2] dataset/sample_zeek_files - port 32769
[3] dataset/sample_zeek_files - port 32770
```

you can select the number you want to kill or 0 to close all the servers.

Note that if all ports from (32768 to 32850) are unavailable, slips won't be able to start, and you will be asked to close all all of them using the following warning

```
All ports from 32768 to 32769 are used. Unable to start slips.

Press Enter to close all ports.
```

You can press enter to close all ports, then start slips again.

2.5 Growing zeek directories

Due to issues running Slips on an interface on MacOS, we added the option `-g` to run slips on a growing zeek directory

so you can run slips in docker, mount a into the container then start zeek inside it using the following command

```
bro -C -i <interface> tcp_inactivity_timeout=60mins tcp_attempt_delay=1min <path_to>/slips/zeek-scripts
```

Then start slips on your zeek dir using -f normally, and mark the given dir as growing using -g

```
./slips.py -e 1 -f zeek_dir/ -g
```

By using the -g parameter, slips will treat your given zeek directory as growing (the same way we treat zeek directories generated by using slips with -i) and will not stop when there are no flows for a while.

2.6 Reading the output

The output process collects output from the modules and handles the display of information on screen. Currently, Slips' analysis and detected malicious behaviour can be analyzed as following:

- **Kalipso** - Node.JS based graphical user interface in the terminal. Kalipso displays Slips detection and analysis in colorful table and graphs, highlighting important detections. See section Kalipso for more explanation.
- **alerts.json and alerts.txt in the output folder** - collects all evidences and detections generated by Slips in a .txt and .json formats.
- **log files in a folder *current-date-time*** - separates the traffic into files according to a profile and timewindow and summarize the traffic according to each profile and timewindow.
- **Web interface** - Node.JS browser based GUI for vewing slips detections, Incoming and ongoing traffic and an organized timeline of flows.

There are two options how to run Kalipso Locally:

2.6.1 Kalipso

You can run Kalipso as a shell script in another terminal using the command:

```
./kalipso.sh
```

In docker, you can open a new terminal inside the slips container and execute `./kalipso.sh`

To open a new terminal inside Slips container first run Slips in one terminal

Now in a new local terminal get the Slips container ID:

```
docker ps
```

Create another terminal of the Slips container using

```
docker exec -it <container_id> bash
```

Now you can run

```
./kalipso.sh
```

and choose the port Slips started on. Slips uses port 6379 by default.

On the right column, you can see a list of all the IPs seen in your traffic.

The traffic of IP is splitted into time windows. each time window is 1h long of traffic.

You can press Enter of any of them to view the list of flows in the timewindow.

You can switch to the flows view in kalipso by pressing TAB, now you can scroll on flows using arrows

On the very top you can see the ASN, the GEO location, and the virustotal score of each IP if available

Check how to setup virustotal in Slips here.

2.6.2 The Web Interface

You can use Slips' web interface by running slips with `-w` or running:

./webinteface.sh

Then navigate to `http://localhost:55000/` from your browser.

Just like kalipso, On the right column, you can see a list of all the IPs seen in your traffic.

The traffic of IP is splitted into time windows. each time window is 1h long of traffic.

IPs and timewindows that are marked in red are considered malicious in Slips.

You can view the traffic of each time window by clicking on it

- The timeline button shows the zeek logs formatted into a human readable format by Slips' timeline module
- The flows button shows the raw flows as seen in the input file, of by zeek in case of running Slips on a PCAP or on you rinterface
- The outgoing button shows flow sent from this IP/profile to other IPs only. it doesn't show traffic sent to the profile.
- The incoming button shows flow sent to this IP/profile to other IPs only. It doesn't show traffic sent from the profile.
- The Alerts button shows the alerts Slips saw for this IP, each alert is a bunch of evidence that the given profile is malicious. Slips decides to block the IP if an alert is generated for it (if running with -p). Clicking on each alert expands the evidence that resulted in the alert.
- The Evidence button shows all the evidence of the timewindow whether they were part of an alert or not.

If you're running slips in docker you will need to add one of the following parameters to docker to be able to use the web interface:

Use the host network by adding `--net=host`, for example:

```
docker run -it --rm --net=host --cap-add=NET_ADMIN --name slips stratosphereips/
↪slips:latest
```

Use port mapping to access the container's port to the host's port by adding `-p 55000:55000`, for example:

```
docker run -it --rm -p 55000:55000 --name slips stratosphereips/slips:latest
```

2.7 Saving the database

Slips uses redis to store analysis information. you can save your analysis for later use by running slips with `-s`,

For example:

`sudo ./slips.py -f dataset/test7-malicious.pcap -s`

Your .rdb saved database will be stored in the default output dir.

Note: If you try to save the same file twice using `-s` the old backup will be overwritten.

You can load it again using `-d`, For example:

`sudo ./slips.py -d redis_backups/hide-and-seek-short.rdb`

And then use `./kalipso` or `./webinterface.sh` and select the entry on port 32850 to view the loaded database.

Note: saving and loading the database requires **root privileges** and is only supported in linux.

This feature isn't supported in docker due to problems with redis in docker.

DISCLAIMER: When saving the database you will see the following warning

```
stop-writes-on-bgsave-error is set to no, information may be lost in the redis backup␣
↪file
```

This configuration is set by slips so that redis will continue working even if redis can't write to dump.rdb.

Your information will be lost only if you're out of space and redis can't write to dump.rdb or if you don't have permissions to write to /var/lib/redis/dump.rdb, otherwise you're fine and the saved database will contain all analyzed flows.

2.8 Whitelisting

Slips allows you to whitelist some pieces of data in order to avoid its processing. In particular, you can whitelist an IP address, a domain, a MAC address or a complete organization. You can choose to whitelist what is going **to** them and what is coming **from** them. You can also choose to whitelist the flows, so they are not processed, or the alerts, so you see the flows but don't receive alerts on them. The idea of whitelisting is to avoid processing any communication to or from these pieces of data, not to avoid any packet that contains that piece of data. For example, if you whitelist the domain slack.com, then a DNS request to the DNS server 1.2.3.4 asking for slack.com will still be shown.

2.8.1 Flows Whitelist

If you whitelist an IP address, Slips will check all flows and see if you are whitelisting to them or from them.

If you whitelist a domain, Slips will check:

- Domains in HTTP Host header
- Domains in the SNI field of TLS flows
- All the Domains in the DNS resolution of IPs (there can be many) (be careful that the resolution takes time, which means that some flows may not be whitelisted because Slips doesn't know they belong to a domain).
- Domains in the CN of the certificates in TLS

If you whitelist an organization, then:

- Every IP is checked against all the known IP ranges of that organization

- Every domain (SNI/HTTP Host/IP Resolution/TLS CN certs) is checked against all the known domains of that organization
- ASNs of every IP are verified against the known ASN of that organization

If you whitelist a MAC address, then:

- The source and destination MAC addresses of all flows are checked against the whitelisted mac address.

2.8.2 Alerts Whitelist

If you whitelist some piece of data not to generate alerts, the process is the following:

- If you whitelisted an IP
  - We check if the source or destination IP of the flow that generated that alert is whitelisted.
  - We check if the content of the alert is related to the IP that is whitelisted.
- If you whitelisted a domain
  - We check if any domain in alerts related to DNS/HTTP Host/SNI is whitelisted.
  - We check also if any domain in the traffic is a subdomain of your whitelisted domain. So if you whitelist 'test.com', we also match 'one.test.com'
- If you whitelisted an organization
  - We check that the ASN of the IP in the alert belongs to that organization.
  - We check that the range of the IP in the alert belongs to that organization.
- If you whitelist a MAC address, then:
  - The source and destination MAC addresses of all flows are checked against the whitelisted mac address.

2.8.3 Tranco whitelist

In order to reduce the number of false positive alerts, Slips uses Tranco whitelist which contains a research-oriented top sites ranking hardened against manipulation here https://tranco-list.eu/

Slips download the top 10k domains from this list and by default and whitelists all evidence and alerts from and to these domains. Slips still shows the flows to and from these IoC.

The tranco list is updated daily by default in Slips, but you can change how often to update it using the `online_whitelist_update_period` key in config/slips.conf.

2.8.4 Whitelisting Example

You can modify the file `whitelist.csv` file with this content:

```
"IoCType","IoCValue","Direction","IgnoreType"
ip,1.2.3.4,both,alerts
domain,google.com,src,flows
domain,apple.com,both,both
ip,94.23.253.72,both,alerts
ip,91.121.83.118,both,alerts
mac,b1:b1:b1:c1:c2:c3,both,alerts
organization,microsoft,both,both
```

(continues on next page)

(continued from previous page)

```
organization,facebook,both,both
organization,google,both,both
organization,apple,both,both
organization,twitter,both,both
```

The values for each column are the following:

```
Column IoCType
    - Supported IoCTypes: ip, domain, organization, mac
Column IoCValue
    - Supported organizations: google, microsoft, apple, facebook, twitter.
Column Direction
    - Direction: src, dst or both
        - Src: Check if the IoCValue is the source
        - Dst: Check if the IoCValue is the destination
        - Both: Check if the IoCValue is the source or destination
Column IgnoreType
    - IgnoreType: alerts or flows or both
        - Ignore alerts: slips reads all the flows, but it just ignores alerting if␣
↪there is a match.
        - Ignore flows: the flow will be completely discarded.
```

2.9 Popup notifications

Slips Support displaying popup notifications whenever there's an alert.

This feature is disabled by default. You can enable it by changing `popup_alerts` to `yes` in `config/slips.conf`

This feature is supported in Linux and it requires root privileges.

This feature is supported in MaOS without root privileges.

This feature is not supported in Docker

2.10 Slips permissions

Slips doesn't need root permissions unless you

1. use the blocking module (with -p)
2. use slips notifications
3. are saving the database (with -d)

If you can't listen to an interface without sudo, you can run the following command to let any user use zeek to listen to an interface not just root.

```
sudo setcap cap_net_raw,cap_net_admin=eip /<path-to-zeek-bin/zeek
```

Even when Slips is run using sudo, it drops root privileges in modules that don't need them.

2.11 Modifying the configuration file

Slips has a `config/slips.conf` the contains user configurations for different modules and general execution. Below are some of Slips features that can be modifie with respect to the user preferences.

2.11.1 Generic configuration

Time window width.

Each IP address that appears in the network traffic of the input is represented as a profile in Slips. Each profile is divided into time windows. Each time window is 1 hour long by default, and it gathers the network traffic and its behaviour for the period of 1 hour. The duration of the timewindow can be changed in the the config/slips.conf using

`time_window_width`

Analysis Direction

`analysis_direction` can either be `out` or `all`

The `analysis_direction` parameter controls which traffic flows are processed and analyzed by SLIPS. It determines whether SLIPS should focus on outbound traffic (potential data exfiltration), inbound traffic (incoming attacks), or analyze traffic in both directions. In `all` mode, SLIPS creates profiles for both internal (A) and external (B) IP addresses, and analyzes traffic in both directions (inbound and outbound).

In `out` mode, SLIPS still creates profiles for internal (A) and external (B) IP addresses, but only analyzes the outgoing traffic from the internal (A) profiles to external (B) destinations.

This parameter allows you to tailor SLIPS's analysis focus based on your specific monitoring requirements, such as detecting potential data exfiltration attempts (`out` mode) or performing comprehensive network monitoring in both directions (`all` mode).

2.11.2 Disabling a module

You can disable modules easily by appending the module name to the `disable` list.

The module name to disable should be the same as the name of it's directory name inside modules/ directory

2.11.3 ML Detection

The `mode=train` should be used to tell the MLdetection1 module that the flows received are all for training.

The `mode=test` should be used after training the models, to test unknown data.

You should have trained at least once with 'Normal' data and once with 'Malicious' data in order for the test to work.

2.11.4 Blocking

This module is enabled only using the `-p` parameter and needs an interface to run.

Usage example:

```
sudo ./slips.py -i wlp3s0 -p
```

Slips needs to be run as root so it can execute iptables commands.

In Docker, since there's no root, the environment variable `IS_IN_A_DOCKER_CONTAINER` should be set to `True` to use 'sudo' properly.

If you use the latest Dockerfile, it will be set by default. If not, you can set it manually by running this command in the docker container

```
export IS_IN_A_DOCKER_CONTAINER=True
```

2.11.5 VirusTotal

In order for virustotal module to work, you need to add your VirusTotal API key to the file `config/vt_api_key`.

You can specify the path to the file with VirusTotal API key in the `api_key_file` variable.

The file should contain the key at the start of the first line, and nothing more.

If no key is found, virustotal module will not start.

2.11.6 Exporting Alerts

Slips can export alerts to different systems.

Refer to the exporting section of the docs for detailed instructions on how to export.

2.12 Logging

To enable the creation of log files, there are two options:

1. Running Slips with `verbose` and `debug` flags
2. Using errors.log and running.log

2.12.1 Zeek log files

You can enable or disable deleting zeek log files after stopping slips by setting `delete_zeek_files` to yes or no.

DISCLAIMER: zeek generates log files that grow every second until they reach GBs, to save disk space, Slips deletes all zeek log files after 1 day when running on an interface. this is called zeek rotation and is enabled by default.

You can disable rotation by setting `rotation` to `no` in `config/slips.conf`

Check rotation section for more info

But you can also enable storing a copy of zeek log files in the output directory after the analysis is done by setting `store_a_copy_of_zeek_files` to yes, or while zeek is stil generating log files by setting `store_zeek_files_in_the_output_dir` to yes. this option stores a copy of the zeek files present in `zeek_files/` the moment slips stops. so this doesn't include deleted zeek logs.

Once slips is done, you will find a copy of your zeek files in `<output_dir>/zeek_files/`

DISCLAIMER: Once slips knows you do not want a copy of zeek log files after slips is done by enabling `delete_zeek_files` and disabling `store_a_copy_of_zeek_files` parameters, it deletes large log files periodically (like arp.log).

2.12.2 Rotation of zeek logs

Rotation is done in zeek files to avoid growing zeek log files and save disk space.

Rotating is only enabled when running on an interface.

By default, Slips rotates zeek files every 1 day. this can be changed in `config/slips.conf` by changing the value of `rotation_period`

`rotation_period` value can be written as a numeric constant followed by a time unit where the time unit is one of usec, msec, sec, min, hr, or day which respectively represent microseconds, milliseconds, seconds, minutes, hours, and days. Whitespace between the numeric constant and time unit is optional. Appending the letter s to the time unit in order to pluralize it is also optional. Check Zeek rotation interval for more details

Slips has an option to not delete the rotated zeek files immediately by setting the `keep_rotated_files_for` parameter.

This is equivalent to telling slips, Delete all logs that happened before the last `keep_rotated_files_for` days.

`keep_rotated_files_for` value supports days only.

2.12.3 Running Slips with verbose and debug flags

We use two variables for logging, `verbose` and `debug`, they both range from 0 to 3.

Default value for debug is 0. so no errors are printed by default.

Default value for verbose is 1. so basic operations and proof of work are printed by default.

To change them, We use `-v` for verbosity and `-e` for debugging

For example:

```
./slips.py -c config/slips.conf -v 2 -e 1 -f zeek_dir
```

Verbosity is about less or more information on the normal work of slips.

For example: "Done writing logs to file x."

Debug is only about errors.

For example: "Error reading threat intelligence file, line 4, column 2"

To more verbosity level, the more detailed info is printed.

The more debug level, the more errors are logged.

Below is a table showing each level of both.

2.12.4 Using errors.log and running.log

Slips also logs all errors to output/errors.log (interactive mode), and /var/log/slips/error.log (daemonized mode) whether you're using -e or not. See Daemonized vs interactive for more information about the 2 modes

General slips logs are created in /var/log/slips/running.log in case of daemonized mode and …#TODO in case of interactive mode

2.13 Reading Input from stdin

Slips supports reading input flows in text format from stdin in interactive mode.

Supported flow from stdin are zeek conn.log files (json form), suricata and argus.

For example, you can run slips using:

`./slips.py -f zeek`

or

`./slips.py -f suricata`

and you once you see the following line:

`[InputProcess] Receiving flows from stdin.`

you can start giving slips flows in you desired format.

All zeek lines taken from stdin should be in json form and are treated as conn.log lines.

This feature is specifically designed to allow slips to interact with network simulators and scripts.

2.14 Plug in a zeek script

Slips supports automatically running a custom zeek script by adding it to `zeek-scripts` dir and adding the file name in `zeek-scripts/__load__.zeek`.

For example, if you want to add a zeek script called `arp.zeek` you should add it to `__load__.zeek` like this:

```
@load ./arp.zeek
```

Zeek output is suppressed by default, so if your script has errors, Slips will fail silently.

2.15 Slips parameters

- `-c` or `--config` Used for changing then path to the Slips config file. default is config/slips.conf
- `-v` or `--verbose` Verbosity level. This logs more info about Slips.
- `-e` or `--debug` Debugging level. This shows more detailed errors.
- `-f` or `--filepath` Read and automatically recognize a Zeek dir, a Zeek conn.log file, a Suricata JSON file, Argus, PCAP.
- `-i` or `--interface` Read packets from an interface.
- `-l` or `--createlogfiles` Create log files with all the traffic info and detections.
- `-F` or `--pcapfilter` Packet filter for Zeek. BPF style.
- `-cc` or `--clearcache` Clear the cache database.
- `-p` or `--blocking` Allow Slips to block malicious IPs. Requires root access. Supported only on Linux.
- `-cb` or `--clearblocking` Flush and delete slipsBlocking iptables chain
- `-o` or `--output` Store alerts.json and alerts.txt in the given folder.
- `-s` or `--save` Save the analysed file db to disk.

- `-d` or `--db` Read an analysed file (rdb) from disk.
- `-D` or `--daemon` Run slips in daemon mode
- `-S` or `--stopdaemon` Stop slips daemon
- `-k` or `--killall` Kill all unused redis servers
- `-m` or `--multiinstance` Run multiple instances of slips, don't overwrite the old one
- `-P` or `--port` The redis-server port to use
- `-g` or `--growing` Treat the given zeek directory as growing. eg. zeek dirs generated when running onan interface
- `-w` or `--webinterface` Start Slips web interface automatically
- `-V` or `--version` Used for checking your running Slips version flags.
- `-im` or `--input-module` Used for reading flows from a module other than input process.

2.16 Containing Slips resource consumption

When given a very a large pcap, slips may use more memory/CPU than it should. to fix that you can reduce the niceness of Slips by running:

```
renice -n 6 -p <Slips-PID>
```

2.17 Running Slips from python

You can run Slips from python using the following script

```python
import subprocess
command = './slips.py -f dataset/test3-mixed.binetflow -o /data/test'
args = command.split()
process = subprocess.run(args, stdout=subprocess.PIPE)
```

CHAPTER

THREE

ARCHITECTURE

The architecture of Slips is basically: - To receive some data as input - To process it to a common format - To enrich it (gather all possible info about the IPs/MAC/User-Agents etc.) - To apply detection modules - To output results

Slips is heavily based on the Zeek monitoring tool as input tool for packets from the interface and pcap file, due to its excelent recognition of protocols and easiness to identify the content of the traffic.

Figure 1 shows how the data is analyzed by Slips. As we can see, Slips internally uses Zeek, an open source network security monitoring tool. Slips divides flows into profiles and each profile into a timewindows. Slips runs detection modules on each flow and stores all evidence, alerts and features in an appropriate profile structure. All profile info, performed detections, profiles and timewindows' data, is stored inside a Redis database. All flows are read, interpreted by Slips, labeled, and stored in the SQLite database in the output/ dir of each run The output of Slips is a folder with logs (output/ directory) that has alert.json, alerts.log, errors.log. Kalipso, a terminal graphical user interface. or the Web interface.

Below is more explanation on internal representation of data, usage of Zeek and usage of Redis inside Slips.

3.1 Internal representation of data.

Slips works at a flow level, instead of a packet level, gaining a high level view of behaviors. Slips creates traffic profiles for each IP that appears in the traffic. A profile contains the complete behavior of an IP address. Each profile is divided into time windows. Each time window is 1 hour long by default and contains dozens of features computed for all connections that start in that time window. Detections are done in each time window, allowing the profile to be marked as uninfected in the next time window.

This is what slips stores for each IP/Profile it creates:

- Ipv4 - ipv4 of this profile
- IPv6 - list of ipv6 used by this profile
- Threat_level - the threat level of this profile, updated every TW.
- Confidence - how confident slips is that the threat level is correct
- Past threat levels - history of past threat levels
- Used software - list of software used by this profile, for example SSH, Browser, etc.
- MAC and MAC Vendor - Ether MAC of the IP and the name of the vendor
- Host-name - the name of the IP
- first User-agent - First UA seen use dby this profile.
- OS Type - Type of OS used by this profile as extracted from the user agent
- OS Name - Name of OS used by this profile as extracted from the user agent

- Browser - Name of the browser used by this profile as extracted from the user agent
- User-agents history - history of the all user agents used by this profile
- DHCP - if the IP is a dhcp or not
- Starttime - epoch formatted timestamp of when the profile first appeared
- Duration - the standard duration of every TW in this profile
- Modules labels - the labels assigned to this profile by each module
- Gateway - if the IP is the gateway (router) of the network
- Timewindow count - Amount of timewindows in this profile
- ASN - autonomous service number of the IP
- Asnorg - name of the org that own the ASN of this IP
- ASN Number
- SNI - Server name indicator
- Reverse DNS - name of the IP in reverse dns
- Threat Intelligence - If the IP appeared in any of Slips blacklist
- Description - Description of this IP as taken from the blacklist
- Blacklist Threat level - threat level of the blacklisted that has this IP
- Passive DNS - All the domains that resolved into this IP
- Certificates - All the certificates that were used by this IP
- Geocountry - Country of this IP
- VirusTotal - contains virustotal scores of this IP
  - Down_file: files in virustotal downloaded from this IP
  - Ref_file: files in VT that referenced this IP
  - Com_file : files in VT communicating with this IP
  - Url ratio: The higher the score the more malicious this IP is

3.2 Alerts vs Evidence

When running Slips, the alerts you see in red in the CLI or at the very bottom in kalispo, are a bunch of evidence. Evidence in slips are detections caused by a specific IP in a specific timeframe. Slips doesn't alert on every evidence/detection. it accumulates evidence and only generates and alert when the amount of gathered evidence crosses a threshold. After this threshold Slips generates an alert, marks the timewindow as malicious(displays it in red in kalipso) and blocks the IP causing the alert.

3.3 Usage of Zeek.

Slips uses Zeek to generate files for most input types, and this data is used to create the profiles. For example, Slips uses this data to create a visual timeline of activities for each time window. This timeline consists of Zeek generated flows and additional interpretation from other logs like dns log and http log.

3.4 Usage of Redis database.

All the data inside Slips is stored in Redis, an in-memory data structure. Redis allows all the modules in Slips to access the data in parallel. Apart from read and write operations, Slips takes advantage of the Redis messaging system called Redis PUB/SUB. Processes may publish data into the channels, while others subscribe to these channels and process the new data when it is published.

3.5 Usage of SQLite database.

Slips uses SQLite database to store all flows in Slips interpreted format. The SQLite database is stored in the output/ dir and each flow is labeled to either 'malicious' or 'benign' based on slips detections. all the labeled flows in the SQLite database can be exported to tsv or json format.

3.6 Threat Levels

Slips has 4 threat levels.

CHAPTER

FOUR

DETECTION MODULES

Slips is a behavioral-based IPS that uses machine learning to detect malicious behaviors in the network traffic. It is a modular software that can be extended. When Slips is run, it spawns several child processes to manage the I/O, to profile attackers and to run the detection modules.

Here we describe what detection modules are run on the traffic to detect malicious behaviour.

Modules are Python-based files that allow any developer to extend the functionality of Slips. They process and analyze data, perform additional detections and store data in Redis for other modules to consume. Currently, Slips has the following modules:

4.1 Virustotal Module

This module is used to lookup IPs, domains, and URLs on virustotal.

To use it you need to add your virustotal API key in `config/vt_api_key`

4.2 RiskIQ Module

This module is used to get different information (passive DNS, IoCs, etc.) from RiskIQ To use this module your RiskIQ email and API key should be stored in `config/RiskIQ_credentials`

the format of this file should be the following:

```
example@domain.com
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

The hash should be your 64 character API Key.

The path of the file can be modified by changing the `RiskIQ_credentials_path` parameter in `config/slips.conf`

4.3 RNN C&C Detection Module

This module is used to detect command and control channels in a network by analyzing the features of the network flows and representing them as Stratosphere Behavioral Letters(Stratoletters). This is achieved through the use of a recurrent neural network, which is trained on these letters to identify and classify potential C&C traffic.

4.3.1 Stratoletters

Stratoletters is a method used to represent network flows in a concise and standardized manner. Stratoletters encodes information about the periodicity, duration, and size of network flows into a string of letter(s) and character(s).

The **letter** is the key part in a Stratoletter string. It is derived from a dictionary and defined based on the features of the flow such as periodicity, size and duration.

periodicity : A number that denotes how frequent the flow is, calculated based on time of past flows

-1 = No previous data

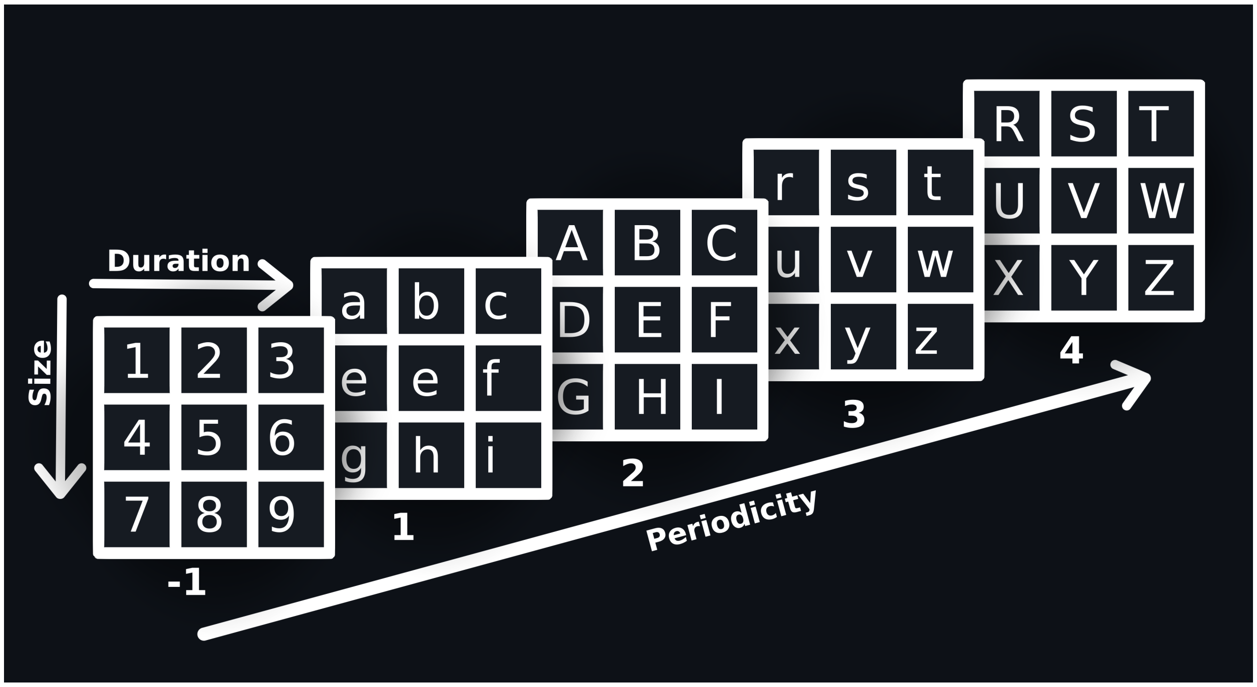
1-4 = 1 strongly periodic to 4 strongly not periodic

size : Number denotes the size of flow, range from 1 to 3

duration : Number that denotes the duration of flow, range from 1 to 3

A visual representation of the dictionary from which letter is derived

In this image, each block represents the possible values of the letter, We choose the block based on the periodicity and choose the letter from the block based on the duration(number of row) and size (number of column).



Example:

```
# Slips computed value of the flow
periodicity = 1     # Strongly periodic
duration = 1
```

(continues on next page)

(continued from previous page)

```
size = 3
letter = g          # lowercase letter for periodicity 1(Strongly periodic) and 3(Weakly␣
↪not periodic)
```

```
periodicity = -1    # no previous flow data
duration = 2
size = 3
letter = 8          # the letter will be an integer if there is no previous data
```

```
periodicity = 4     # Weakly not periodic
duration = 3
size = 3
letter = Z          # uppercase letter for periodicity 2(Weakly periodicity) and␣
↪4(Strongly not periodicity)
```

Stratoletters represent details of current flow and past available flow with latest on left. The current flow symbol is concise of three parts.

each symbol consists of hrs passed since last flow + a letter that represents the periodicity,size and dur of the flow + a char showing the time passed since last flow

```
symbol = zeros + letter + timechar
```

zero : hours passed since last flow, each hour is represented by 1 zero. foe xample, 2 hours = `00`

letter : chosen based on the periodicity, size, and dur of the flow eg: `1,w,H`

timechar : character to denote the time eloped since last flow, can be: `.`, `,`, +, * or null

Ultimately this is how a Stratoletter is formed

```
No of hours passed since last flow = 2
periodicity = 2     # Weakly not periodicity
duration = 1
size = 1
timechar =
stratoletter of last flow = 9*z*
letter = A
symbol = 00A9*z*

No of hours passed since last flow = 0
periodicity = 3     # Weakly not periodic
duration = 1
size = 2
timechar = *
stratoletter of last flow = e.
letter = u
symbol = u*e.
```

Then the model will predict how secure each flow is based on the Stratoletter

```
symbol = 99*z*i.i*
model_score = 0.9573354
symbol = 99.
```

(continues on next page)

(continued from previous page)

```
model_score = 0.9063127
symbol = 77*g.g*g*g.g.g.g*x*x*x*g.g.
model_score = 0.96772265
```

In first example **9** is Stratoletter of current flow. **9*** is previous one, **z*** is before that and so on.

4.4 Leak Detection Module

This module on runs on pcaps, it uses YARA rules to detect leaks.

You can add your own YARA rule in `modules/leak_detector/yara_rules/rules` and it will be automatically compiled and stored in `modules/leak_detector/yara_rules/compiled` and matched against every pcap.

4.5 Blocking Module

To enable blocking in slips, start slips with the `-p` flag.

This feature is only supported in linux using iptables.

4.6 Exporting Alerts Module

Slips supports exporting alerts to other systems using different modules (ExportingAlerts, CESNET sharing etc.)

For now the supported systems are:

- Slack
- TAXII Servers (STIX format)
- Warden servers
- suricata-like JSON format
- Logstash

Refer to the exporting section of the docs for detailed instructions on how to export.

4.7 Flowalerts Module

This module is responsible for detecting malicious behaviours in your traffic.

Refer to the Flowalerts section of the docs for detailed explanation of what Slips detects and how it detects.

4.8 Disabled alerts

All Slips detections are turned on by default, You can configure which alerts you want to enable/disable in `config/slips.conf`

Slips support disabling unwanted alerts, simply add the detection you want to disable in the `disabled_detections` list and slips will not generate any alerts of this type.

for example:

```
disabled_detections = [MaliciousJA3, DataExfiltration, SelfSignedCertificate]
```

Supported detections are:

ARPScan, ARP-outside-localnet, UnsolicitedARP, MITM-ARP-attack, SSHSuccessful, LongConnection, MultipleReconnectionAttempts, ConnectionToMultiplePorts, InvalidCertificate, UnknownPort, Port0Connection, ConnectionWithoutDNS, DNSWithoutConnection, MaliciousJA3, DataExfiltration, SelfSignedCertificate, VerticalPortscan, HorizontalPortscan, Password_Guessing, MaliciousFlow, SuspiciousUserAgent, multiple_google_connections, NETWORK_gps_location_leaked, Command-and-Control-channels-detection, ThreatIntelligenceBlacklistDomain, ThreatIntelligenceBlacklistIP, MaliciousDownloadedFile, DGA, MaliciousSSLCert, YoungDomain, MultipleSSHVersions DNS-ARPA-Scan, SMTPLoginBruteforce, BadSMTPLogin, IncompatibleUserAgent, ICMP-Timestamp-Scan, ICMP-AddressScan, ICMP-AddressMaskScan

4.9 Threat Intelligence Module

Slips has a complex system to deal with Threat Intelligence feeds. The Threat Intelligence Module in Slips is designed to enhance detection capabilities, utilizing both external and internal thread intelligence sources.

Slips supports various indicators of compromise (IoCs) from Threat Intelligence (TI) feeds, including IPs, IP ranges, domains, JA3 hashes, and SSL hashes. This provides comprehensive coverage against potential threats

While file hashes and URLs aren't supported in TI feeds directly, Slips compensates by integrating with specialized external services for these types of IoCs.

4.9.1 External Threat Intelligence Services

Besides searching 40+ TI files for every IP/domain Slips encounters, Slips integrates with the following external threat intelligence services to enrich its detection capabilities:

1. **URLhaus**: This service is utilized for checking URLs observed in `http.log` and files observed in `files.log` against known malicious URLs and files. URLhaus provides a comprehensive database of malicious URLs, which Slips queries to determine if observed URLs or files are associated with known malware or phishing campaigns.
2. **Spamhaus**: Spamhaus is used for IP lookups to assess the reputation of IP addresses encountered during the analysis. By querying Spamhaus, Slips can identify IP addresses associated with spamming activities, botnets, and other malicious behaviors, enhancing its ability to detect and alert on suspicious network traffic.
3. **Circl.lu**: Circl.lu's service is leveraged for hash lookups, particularly for downloaded files. Each file hash extracted from `files.log` is checked against Circl.lu's extensive database of known malicious file hashes. This integration allows Slips to identify and react to the transfer or presence of known malicious files within the monitored network environment.

URLhaus Access:

- **Purpose**: Identify malicious URLs and files.

- **Method**: Slips queries the URLhaus API with URLs and file hashes observed in network traffic logs.
- **Response Handling**: If a URL or file is found in the URLhaus database, Slips generates an alert indicating the presence of a known threat.

Spamhaus Access:

- **Purpose**: Assess the reputation of IP addresses.
- **Method**: IP addresses are queried against Spamhaus's DNSBL (DNS-based Block List).
- **Response Handling**: Slips interprets the DNSBL response to determine if an IP address is associated with known malicious activities, triggering alerts accordingly.

Circl.lu Access:

- **Purpose**: Perform hash lookups for downloaded files.
- **Method**: File hashes are checked against Circl.lu's database via their API.
- **Response Handling**: Matches with known malicious hashes result in the generation of alerts to inform about potential threats.

By integrating these external services, Slips significantly enhances its detection capabilities, allowing for real-time alerting on threats identified through global intelligence feeds. This integration not only broadens the scope of detectable threats but also contributes to the overall security posture by enabling proactive responses to emerging threats.

4.9.2 Matching of IPs

Slips gets every IP it can find in the network (DNS answers, HTTP destination IPs, SSH destination IPs, etc.) and tries to see if it is in any blacklist.

If a match is found, it generates an evidence, if no exact match is found, it searches the Blacklisted ranges taken from different TI feeds.

4.9.3 Matching of Domains

Slips gets every domain that can find in the network and tries to see if it is in any blacklist. The domains are currently taken from:

- DNS requests
- DNS responses
- HTTP host names
- TLS SNI

Once a domain is found, it is verified against the downloaded list of domains from the blacklists defined in `ti_files` path in the configuration file `config/slips.conf`. which is `config/TI_feeds.csv` by default. If an exact match is found, then an evidence is generated.

If an exact match is not found, then Slips verifies if the verified domain is a subdomain of any domain in the blacklist.

For example, if the domain in the traffic is *here.testing.com*, Slips first checks if the exact domain *here.testing.com* is in any blacklist, and if there is no match, it checks if the domain *testing.com* is in any blacklists too.

4.9.4 Matching of JA3 Hashes

Every time Slips encounters an TLS flow, it compares each JA3 and JA3s with the feeds of malicious JA3 and alerts when there's a match. Slips is shipped with the Abuse.ch JA3 feed by default You can add your own SSL feed by appending to the `ja3_feeds` key in `config/slips.conf`

4.9.5 Matching of SSL SHA1 Hashes

Every time Slips encounters an SSL flow, it tries to get the certificate hash from zeek ssl.log, then it compares the hash with our list of blacklisted SSL certificates

Slips is shipped with the Abuse.ch SSL feed by default,

You can add your own SSL feed by appending to the `ssl_feeds` key in `config/slips.conf`

4.9.6 Matching of ASNs

Every time Slips sees a new IP, it stores info about it in the db, for example its organization, RDNs, and ASN. If the ASN of an IP matches a blacklisted ASN, slips alerts.

Blacklisted ASNs are read from out local TI file `config/local_data_files/own_malicious_iocs.csv`, so you can update them or add your own.

4.9.7 Local Threat Intelligence files

Slips has a local file for adding IoCs of your own, it's located in `config/local_data_files/own_malicious_iocs.csv` by default, this path can be changed by changing `download_path_for_local_threat_intelligence` in `config/slips.conf`.

The format of the file is "IP address/IP Range/domain/ASN","Threat level", "Description"

Threat level available options: info, low, medium, high, critical

Refer to the architecture section of the docs for detailed explanation of Slips threat levels.

Example:

```
"23.253.126.58","high","Simda CC"
"bncv00.no-ip.info", "critical", "Variant.Zusy"
```

4.9.8 Local JA3 hashes

Slips has a local file for adding JA3 hashes of your own, it's located in `config/local_data_files/own_malicious_JA3.csv` by default.

The format of the file is "JA3 hash", "Threat level", "Description"

Threat level available options: info, low, medium, high, critical

Refer to the architecture section of the docs for detailed explanation of Slips threat levels.

Example:

```
"e7d705a3286e19ea42f587b344ee6865","medium","Standard tor client"
"6734f37431670b3ab4292b8f60f29984", "high", "Trickbot Malwar"
```

4.9.9 Adding your own remote feed

We update the remote ones regularly. The list of remote threat intelligence files is set in the path of `ti_files` variable in `config/slips.conf`. The path of all the TI feeds is in `config/TI_feeds.csv` by default.

You can add your own remote threat intelligence feeds in this variable. Supported extensions are: .txt, .csv, .netset, ipsum feeds, or .intel.

Each URL should be added with a threat_level and a tag, the format is (url,threat_level,tag)

tag is which category is this feed e.g. phishing, adtrackers, etc..

Threat level available options: info, low, medium, high, critical

Refer to the architecture section of the docs for detailed explanation of Slips threat levels.

TI files commented using # may be processed as they're still in our database.

Use `;` for commenting TI files in `config/slips.conf` instead of `#`.

Commented TI files (lines starting with ;) will be completely removed from our database.

The remote files are downloaded to the path set in the `download_path_for_local_threat_intelligence`. By default, the files are stored in the Slips directory `modules/ThreatIntelligence1/remote_data_files/` are deleted after slips is done reading them.

4.9.10 Commenting a remote TI feed

If you have a remote file link that you wish to comment and remove from the database you can do so by adding ';' to the line that contains the feed link in `config/TI_feeds.csv`, don't use the '#' for example to comment the `bruteforcelogin` feed you should do the following:

```
;https://lists.blocklist.de/lists/bruteforcelogin.txt, medium,['honeypot']
```

instead of:

```
#https://lists.blocklist.de/lists/bruteforcelogin.txt,medium,['honeypot']
```

4.10 Update Manager Module

To make sure Slips is up to date with the most recent IoCs in all feeds, all feeds are loaded, parsed and updated periodically and automatically by Slips every 24 hours, which requires no user interaction.

The 24 hours interval can be changed by changing the `TI_files_update_period` key in `config/slips.conf`

Update manager is responsible for updating all remote TI files (including SSL and JA3 etc.)

By default, local slips files (organization_info, ports_info, etc.) are cached to avoid loading and parsing

then everytime we start slips. However, they are updated automatically by the update manager if they were changed on disk.

Only one slips instance is allowed to be using the update manager at a time to avoid race conditions.

By default, slips starts without the TI files, and runs the Update Manager in the background if the `wait_for_TI_to_finish` option in slips.conf is set to yes, slips will not start until the update manager is done and all TI files are loaded successfully, this is useful if you want to ensure that slips doesn't miss the detection of any blacklisted IPs, but it adds some time to the startup of slips since it will be downloading, parsing, and caching 45+ different TI feeds.

4.11 IP Info Module

The IP info module has several ways of getting information about an IP address, it includes:

- ASN
- Country by Geolocation
- Given a MAC, its Vendor
- Reverse DNS

4.11.1 ASN

Slips is shipped with an offline database (GeoLite2) in `databases/GeoLite2-ASN.mmdb` to search for ASNs, if the ASN of a given IP is not in the GeoLite2 database, we try to get the ASN online using the online database using the `ipwhois` library. However, to reduce the amount of requests, we retrieve the range of the IP and we cache the whole range. To search and cache the whole range of an IP, the module uses the ipwhois library. The ipwhois library gets the range of this IP by making a connection to the server `cymru.com` using a TXT DNS query. The DNS server is the one set up in the operating system. For example to get the ASN of the IP 13.32.98.150, you will see a DNS connection asking for the TXT record of the domain `150.98.32.13.origin.asn.cymru.com`.

4.11.2 Country by Geolocation

Slips is shipped with an offline database (GeoLite2) in `databases/GeoLite2-Country.mmdb` to search for Geolocation.

4.11.3 Mac Vendors

Slips is shipped with an offline database `databases/macaddress-db.json` for MAC address vendor mapping.

Slips updates this database by default every 2 weeks using the following online db

https://maclookup.app/downloads/json-database/get-db?t=22-08-19&h=d1d39c52de447a7e7194331f379e1e99f94f35f1

You can change how often this db is updated by changing the value of `mac_db_update` in `config/slips.conf`.

Slips gets the MAC address of each IP from dhcp.log and arp.log and then searches the offline database using the OUI.

If the vendor isn't found in the offline MAC database, Slips tries to get the MAc using the online database https://www.macvendorlookup.com

The offline database is updated manually and shipped with slips, you can find it in the `databases/` dir.

Slips makes sure it doesn't perform duplicate searches of the same MAC Address either online, or offline.

4.12 Reverse DNS

This is obtained by doing a standard in-addr.arpa DNS request.

4.13 ARP Module

This module is used to check for ARP attacks in your network traffic.

By default, zeek doesn't generate and log ARP flows, but Slips is shipped with it's own zeek scripts that enable the logging of ARP flows in `arp.log`

The detection techniques are:

- ARP scans
- ARP to a destination IP outside of local network
- Unsolicited ARP
- MITM ARP attack

4.13.1 ARP Scans

Slips considers an IP performing an ARP scan if it sends 5 or more non-gratuitous ARP to different destination addresses in 30 seconds or less.

4.13.2 ARP to a destination IP outside of local network

Slips alerts when an ARP flow is being sent to an IP outside of local network as it's a weird behaviour that shouldn't be happening.

4.13.3 Unsolicited ARP

Unsolicited ARP is used to update the neighbours' ARP caches but can also be used in ARP spoofing, we detect it with threat level 'info', so we don't consider it malicious, we simply notify you about it.

4.13.4 MITM ARP attack

Slips detects when a MAC with IP A, is trying to tell others that now that MAC is also for IP B (ARP cache attack)

4.14 CESNET sharing Module

This module is responsible for importing and exporting alerts from and to warden server

Refer to the exporting section of the docs for detailed instructions on CESNET exporting and the format of the configuration files.

To enable the importing alerts from warden servers, set `receive_alerts` to `yes` in config/slips.conf

Slips imports 100 alerts from warden servers each day, and automatically stores the IoCs in our database

Time to wait before receiving alerts from warden server is 1 day by default, you can change this by chaning the `receive_delay` in `config/slips.conf`

These are the categories Slips imports: [’Availability’, ‘Abusive.Spam’,’Attempt.Login’, ‘Attempt’, ‘Information’, ‘Fraud.Scam’, ‘Information’, ‘Fraud.Scam’]

4.15 HTTP Analyzer Module

This module handles the detections of HTTP flows

Available detection are:

- Multiple empty connections
- Suspicious user agents
- Incompatible user agents
- Multiple user agents
- Pastebin downloads
- Unencrypted HTTP traffic

4.15.1 Multiple empty connections

Due to the usage of empty connections to popular site by malware to check for internet connectivity, We consider this type of behaviour suspicious activity that shouldn’t happen

We detect empty connection to ‘bing.com’, ‘google.com’, ‘yandex.com’, ‘yahoo.com’ , ‘duckduckgo.com’ etc.

4.15.2 Suspicious user agents

Slips has a list of suspicious user agents, whenever one of them is found in the traffic, slips generates and evidence.

Our current list of user agents has: [’httpsend’, ‘chm_msdn’, ‘pb’, ‘jndi’, ‘tesseract’]

4.15.3 Incompatible user agents

Slips uses and offline MAC address database to detect the type of device based on the MAC OUI.

First, Slips store the MAC address and vendor of every IP it sees (if available)

Second, When slips encounters a user agent in HTTP traffic it performs an online query to http://useragentstring.com to get more info about this user agent, like the os type, name and browser.

Third, When slips has both information available (MAC vendor and user agent), it compares them to detect incompatibility using a list of keywords for each operating system.

Available keywords for Apple: (’macos’, ‘ios’, ‘apple’, ‘os x’, ‘mac’, ‘macintosh’, ‘darwin’)

Available keywords for Microsoft: (’microsoft’, ‘windows’, ‘nt’)

Available keywords for Android: (’android’, ‘google’)

4.15.4 Multiple user agents

Slips stores the MAC address and vendor of every IP it sees (if available) in the redis database. Then, when an IP iss seen using a different user agent than the one stored in the database, it tries to extract os info from the user agent string, either by performing an online query to http://useragentstring.com or by using zeek.

If an IP is detected using different user agents that refer to different operating systems, an alert of type 'Multiple user agents' is made

for example, if an IP is detected using a macOS user agent then an android user agent, slips detects this with 'low' threat level

4.15.5 Pastebin downloads

Some malware use pastebin as the host of their malicious payloads.

Slips detects downloads of files from pastebin through HTTP with size >= 700 bytes.

This value can be customized in slips.conf by changing `pastebin_download_threshold`

When found, slips alerts pastebin download with threat level low because not all downloads from pastebin are malicious.

4.15.6 Unencrypted HTTP traffic

When slip sees an HTTP unencrypted traffic in zeek's http.log it generates an evidence with threat_level low

4.16 Leak Detector Module

This module work only when slips is given a PCAP

The leak detector module uses YARA rules to detect leaks in PCAPs

4.16.1 Module requirements

In order for this module to run you need:

using `sudo apt install yara`

You can install tshark by running

`sudo apt install wireshark`

4.16.2 How it works

This module works by

1. Compiling the YARA rules in the `modules/leak_detector/yara_rules/rules/` directory
2. Saving the compiled rules in `modules/leak_detector/yara_rules/compiled/`
3. Running the compiled rules on the given PCAP
4. Once we find a match, we get the packet containing this match and set evidence.

4.16.3 Extending

You can extend the module be adding more YARA rules in `modules/leak_detector/yara_rules/rules/`.

The rules will be automatically detected, compiled and run on the given PCAP.

If you want to contribute, improve existing Slips detection modules or implement your own detection modules, see section :doc:`Contributing <contributing>`.

4.17 Network Service Discovery Module

This module is responsible for detecting scans such as:

- Vertical port scans
- Horizontal port scans
- PING sweeps
- DHCP Scans

4.17.1 Vertical port scans

Slips considers an IP performing a vertical port scan if it contacts 5 or more different destination ports to the same destination IP in at least one time window (usually 1hs). The flows can be both, Non-Established TCP or UDP flows. On each arriving flow this check is performed.

After detecting a vertical port scan for the first time, if Slips detects new flows to 5 destination ports, then it triggers a waiting process to find out how many packets to new ports will arrive. For this it waits 10 seconds to see if more flows arrive, since in most port scans the attcker will scan more ports. This avoids generating one alert 'port scan' per flow in a long scan. Therfore Slips will wait until the scan finishes to alert on it. However, the first portscan is detected as soon as it happens so the analysts knows.

If one alert was generated (Slips waited 10 seconds and no more flows arrived to new ports in that dst IP) then the counter resets and the same attacker needs to do *again* more than threshold destinations ports in one IP to be detected. This avoids the problem that after 5 flows that generated an alert, the 6 flow also generates an alert.

The total number of *packets* in all flows in the scan give us the confidence of the scan.

4.17.2 Horizontal port scans

Slips considers an IP performing a horizontal port scan if it contacted more than 6 destination IPs on the same specific port with not established connections. Slips checks both TCP and UDP connections for horizontal port scans. The initial threshold is now 6 destination IPs using the same destination ports.

After detecting a horizontal port scan, Slips waits 10 seconds to see if more flows arrive, since in most port scans the attcker will scan more ports. This avoids generating one port scan alert per flow in a long scan. Therfore Slips will wait until the scan finishes to alert on it. However, the first portscan is detected as soon as it happens so the analysts knows.

If one alert was generated (Slips waited 10 seconds and no more flows arrived to new IPs) then the counter resets and the same attacker needs to do *again* more than threshold destinations IPs in the same port to be detected. This avoids the problem that after 6 flows that generated an alert, the 7 flow also generates an alert.

Slips ignores the broadcast IP 255.255.255.255 has destination of port scans.

4.17.3 PING Sweeps

ICMP messages can be used to find out which hosts are alive in a network. Slips relies on Zeek detections for this, but it is done with our own Zeek scripts located in zeek-scripts/icmps-scans.zeek. The scripts detects three types of ICMP scans: 'ICMP-Timestamp', 'ICMP-Address', 'ICMP-AddressMask'.

We detect a scan every threshold. So we generate an evidence when there is 5,10,15, .. etc. ICMP established connections to different IPs.

Slips does this detection using Slips' own zeek script located in zeek-scripts/icmps-scans.zeek for zeek and pcap files and using the portscan module for binetflow files.

4.17.4 DHCP Scans

DHCP requests can be used to find out which IPs are taken in a network. Slips detects when an IP is requesting 4, 8, 12, etc. different IPs from the DHCP server within the same twimewindow (1 hour by default)

CHAPTER

FIVE

CONNECTIONS MADE BY SLIPS

Slips uses online databases to query information about many different things, for example (user agents, mac vendors etc.)

The list below contains all connections made by Slips

useragentstring.com -> For getting user agent info if no info was found in Zeek macvendorlookup.com -> For getting MAC vendor info if no info was found in the local maxmind db maclookup.app -> For getting MAC vendor info if no info was found in the local maxmind db ip-api.com -> For getting ASN info about IPs if no info was found in our Redis DB ipinfo.io -> For getting your public IP virustotal.com -> For getting scores about domains, IPs and URLs urlhaus-api.abuse.ch -> For getting info about URLs and downloaded files check.torproject.org -> For getting info about tor exist nodes. cert.pl -> Used in our list of TI files. abuse.ch -> Used by urlhaus for getting info about contacted domains and downloaded files.

If you want to contribute: improve existing Slips detection modules or implement your own detection modules, see section :doc:`Contributing <contributing>`.

5.1 Zeek Scripts

Slips is shipped with it's own custom zeek scripts to be able to extend zeek functionality and customize the detections

5.1.1 Detect DoH

In the `detect_DoH.zeek` script, slips has it's own list of ips that belong to dns/doh servers,

When slips encouters a connection to any IP of that list on port 443/tcp, it assumes it's a DoH connetion,

and times out the connection after 1h so that the connection won't take too long to appear in slips.

5.1.2 Detect ICMP Scans

In the `zeek-scripts/icmps-scans.zeek` script, we check the type of ICMP in every ICMP packet seen in the network,

and we detect 3 types of ICMP scans: ICMP-Timestamp-Scan, ICMP-AddressScan, and ICMP-AddressMaskScan based on the icmp type

We detect a scan every threshold. So we generate an evidence when there is 5,10,15, .. etc. ICMP established connections to different IPs.

5.1.3 Detect the Gateway address

The `zeek-scripts/log_gw.zeek` script is responsible for recognizing the gateway address using zeek, and logging it to notice.log

CHAPTER

SIX

FLOW ALERTS

The module of flow alerts has several behavioral techniques to detect attacks by analyzing the content of each flow alone.

The detection techniques are:

- Long connections
- Successful SSH connections
- Connections without DNS resolution
- DNS resolutions to IPs that were never used
- Connections to unknown ports
- Data exfiltration
- Malicious JA3 hashes
- Connections to port 0
- Multiple reconnection attempts
- Alerts from Zeek: Self-signed certs, invalid certs, port-scans and address scans, and password guessing
- DGA
- Connection to multiple ports
- Malicious SSL certificates
- Pastebin downloads
- Young domains
- Bad SMTP logins
- SMTP login bruteforce
- DNS ARPA Scans
- SSH version changing
- Incompatible CN
- Weird HTTP methods
- Non-SSL connections on port 443
- Non-HTTP connections on port 80
- Connection to private IPs
- Connection to private IPs outside the current local network

- High entropy DNS TXT answers
- Devices changing IPs
- GRE tunnels
- Invalid DNS answers The details of each detection follows.

6.1 Long Connections

Detect connections that are long, except the multicast connections.

By defualt, A connection in considered long if it exceeds 1500 seconds (25 Minutes).

This threshold can be changed `slips.conf` by changing the value of `long_connection_threshold`

6.2 Connections without DNS resolution

This will detect connections done without a previous DNS resolution. The idea is that a connection without a DNS resolution is slightly suspicious.

If Slips runs by capturing packets directly from a network device (as opposed to, for example, a PCAP file), this detection will ignore all connections that happen in the first 3 minute of operation of Slips. This is because most times Slips is started when the computer is already running, and many DNS connections were already done. So waiting 3 minutes decreases the amount of False Positives.

This detection will ignore certain IP addresses for which a connection without DNS is ok. The exceptions are:

- Private IPs
- Localhost IPs (127.0.0.0/8)
- Reserved IPs (including the super broadcast 255.255.255.255)
- IPv6 local-link IPs
- Multicast IPs
- Broadcast IPs only if they are private
- Well known organizations

DNS resolutions of well known orgs might be done using DoH, in this case, slips doesn't know about the DNS resolution because the resolved domain won't be in dns.log so we simply ignore alerts of this type when connected to well known organizations. In particular Facebook, Apple, Google, Twitter, and Microsoft.

Slips uses it's own lists of organizations and information about them (IPs, IP ranges, domains, and ASNs). They are stored in `slips_files/organizations_info` and they are used to check whether the IP/domain of each flow belong to a known org or not.

Slips doesn't detect 'connection without DNS' when running on an interface except for when it's done by this instance's own IP.

check DoH section of the docs for info on how slips detects DoH.

6.3 Successful SSH connections

Slips detects successful SSH connections using 2 ways

1. Using Zeek. Zeek logs successful SSH connection to ssh.log by default
2. If all bytes sent in a SSH connection is more than 4290 bytes

6.4 DNS resolutions without a connection

This will detect DNS resolutions for which no further connection was done. A resolution without a usage is slightly suspicious.

The domains that are excepted are:

- All reverse DNS resolutions using the in-addr.arpa domain.
- All .local domains
- The wild card domain *
- Subdomains of cymru.com, since it is used by the ipwhois library to get the ASN of an IP and its range.
- Ignore WPAD domain from Windows
- Ignore domains without a TLD such as the Chrome test domains.

6.5 Connection to unknown ports

Slips has a list of known ports located in `slips_files/ports_info/ports_used_by_specific_orgs.csv`

It also has a list of ports that belong to a specific organization in `slips_files/ports_info/ports_used_by_specific_orgs.csv`

For example, even though 5223/TCP isn't a well known port, Apple uses it in Apple Push Notification Service (APNS).

any port that isn't in the above 2 files is considered unknown to Slips.

Slips will detect established connections only to unknown ports.

Rejected connections (not established) are detected as 'Multiple reconnection attempts'. for more details check Multiple reconnections below

6.6 Data Upload

Slips generates 'possible data upload' alerts when the number of uploaded bytes to any IP exceeds 100 MBs over the timewindow period which is, by default, 1h.

See detailed explanation of timewindows here.

The number of MBs can be modified by changing the value of `data_exfiltration_threshold` in `slips.conf`

Slips also detects data upload when an IP uploads >=100MBs to any IP in 1 connections.

6.7 Malicious JA3 and JA3s hashes

Slips uses JA3 hashes to detect C&C servers (JA3s) and infected clients (JA3)

Slips is shipped with it's own zeek scripts that add JA3 and JA3s fingerprints to the SSL log files generated by zeek.

Slips supports JA3 feeds in addition to having more than 40 different threat intelligence feeds. The JA3 feeds contain JA3 fingerprints that are identified as malicious. The JA3 threat intelligence feed used by Slips now is Abuse.ch JA3 feed. And you can add other JA3 TI feeds in `ja3_feeds` in `slips.conf`.

6.8 Connections to port 0

There has been a significant rise in the number of attacks listed as Port 0. Last year, these equated to 10% of all attacks, but now it's up to almost 25%.

Slips detects any connection to port 0 using any protocol other than 'IGMP' and 'ICMP' as malicious.

6.9 Multiple reconnection attempts

Multiple reconnection attempts in Slips are 5 or more not established flows (reconnections) to the same destination IP on the same destination port.

6.10 Zeek alerts

By default, Slips depends on Zeek for detecting different behaviours, for example Self-signed certs, invalid certs, port-scans, address scans, and password guessing.

Password guessing is detected by zeek when 30 failed ssh logins happen over 30 mins.

Some scans are also detected by Slips independently of Zeek, like ICMP sweeps and vertical/horizontal portscans. Check PING Sweeps section for more info

6.11 SMTP login bruteforce

Slips alerts when 3+ invalid SMTP login attempts occurs within 10s

6.12 Password Guessing

Password guessing is detected using 2 ethods in slips

1. by Zeek engine. when 30 failed ssh logins happen over 30 mins.
2. By slips. when 20 failed ssh logins happen over 1 tiemwindow.

6.13 DGA

When the DNS server fails to resolve a domain, it responds back with NXDOMAIN code.

To detect DGA, Slips will count the amount of NXDOMAINs met in the DNS traffic of each source IP.

Then we alert when there is 10 or more NXDOMAINs.

Every 10,15,20 ..etc slips generates an evidence.

6.14 Connection to multiple ports

When Slips encounters a connection to or from a specific IP and a specific port, it scans previous connections looking for connection to/from that same IP using a different port.

It alerts when finding two or more connections to the same IP.

6.15 Malicious SSL certificates

Slips uses SSL certificates sha1 hashes to detect C&C servers.

Slips supports SSL feeds and is shipped with Abuse.ch feed of malicious SSL hashes by default. And you can add other SSL feeds in `ssl_feeds` in `slips.conf`.

6.16 Pastebin downloads

Slips detects downloads from pastebin using SSL and HTTP

It alerts when a downloaded file from pastebin exceeds 700 bytes

This value can be customized in slips.conf by changing `pastebin_download_threshold`

Slips detects the pastebin download once the SSL connection is over , which may take hours.

6.17 Young Domains

Slips uses whois python library to get the creation date of every domain met in the dns flows.

If a domain's age is less than 60 days, slips sets an alert.

Not all domains are supported, here's the list of supported TLDs.

```
['.ac_uk', '.am', '.amsterdam', '.ar', '.at', '.au',
'.bank', '.be', '.biz', '.br', '.by', '.ca', '.cc',
'.cl', '.club', '.cn', '.co', '.co_il', '.co_jp', '.com',
'.com_au', '.com_tr', '.cr', '.cz', '.de', '.download', '.edu',
'.education', '.eu', '.fi', '.fm', '.fr', '.frl', '.game', '.global_',
'.hk', '.id_', '.ie', '.im', '.in_', '.info', '.ink', '.io',
'.ir', '.is_', '.it', '.jp', '.kr', '.kz', '.link', '.lt', '.lv',
'.me', '.mobi', '.mu', '.mx', '.name', '.net', '.ninja',
'.nl', '.nu', '.nyc', '.nz', '.online', '.org', '.pe',
```

(continues on next page)

(continued from previous page)

```
'.pharmacy', '.pl', '.press', '.pro', '.pt', '.pub', '.pw',
'.rest', '.ru', '.ru_rf', '.rw', '.sale', '.se', '.security',
'.sh', '.site', '.space', '.store', '.tech', '.tel', '.theatre',
'.tickets', '.trade', '.tv', '.ua', '.uk', '.us', '.uz', '.video',
'.website', '.wiki', '.work', '.xyz', '.za']
```

6.18 Bad SMTP logins

Slips uses zeek to detect SMTP connections, When zeek detects a bad smtp login, it logs it to smtp.log, then slips reads this file and sets an evidence.

6.19 SMTP bruteforce

Slips detects a SMTP bruteforce when 3 or more bad SMTP logins happen within 10 seconds.

With every generated evidence, Slips gathers as much info about the malicious IP and prints it with the alert.

So instead of having an alerts saying:

```
Detected SSL certificate validation failed with (certificate has expired) Destination␣
↪IP: 216.58.201.70.
```

Slips gathers AS, hostname, SNI, rDNS and any available data about this IP and you get an alert saying:

```
Detected SSL certificate validation failed with (certificate has expired) Destination IP:
216.58.201.70. AS: GOOGLE, US, SNI: 2542116.fls.doubleclick.net, rDNS: prg03s01-in-f70.
↪1e100.net
```

6.20 DNS ARPA Scans

Whenever slips sees a new domain in dns.log, if the domain ends with '.in-addr.arpa' slips keeps trach of this domain and the source IP that made the DNS request.

Then, if the source IP is seen doing 10 or more ARPA queries within 2 seconds, slips generates an ARPA scan detection.

6.21 SSH version changing

Zeek logs the used software and software versions in software.log, so slips knows from this file the software used by different IPs, like whether it's an SSH::CLIENT, an HTTP::BROWSER, or an HTTP::SERVER

When slips detects an SSH client or an SSH server, it stores it with the IP and the SSH versions used in the database

Then whenever slips sees the same IP using another SSH version, it compares the stored SSH versions with the current SSH versions

If they are different, slips generates an alert

6.22 Incompatible CN

Zeek logs each Certificate CN in ssl.log

When slips enccounters a cn that claims to belong to any of Slips supported orgs (Google, Microsoft, Apple or Twitter) Slips checks if the destination address or the destination server name belongs to these org.

If not, slips generates an alert.

6.23 Weird HTTP methods

Slips uses zeek's weird.log where zeek logs weird HTTP methods seen in http.log

When there's a weird HTTP method, slips detects it as well.

6.24 Non-SSL connections on port 443

Slips detects established connections on port 443 that are not using HTTP using zeek's conn.log flows

if slips finds a flow using destination port 443 and the 'service' field in conn.log isn't set to 'ssl', it alerts

6.25 Non-HTTP connections on port 80.

Slips detects established connections on port 80 that are not using SSL using zeek's conn.log flows

if slips finds a flow using destination port 80 and the 'service' field in conn.log isn't set to 'http', it alerts

6.26 Connection to private IPs

Slips detects when a private IP is connected to another private IP with threat level info.

But it skips this alert when it's a DNS connection on port 53 UDP to the gateway

6.27 Connection to private IPs outside the current local network

Slips detects the currently used local network and alerts if it find a connection to/from a private IP that doesn't belong to it.

For example if the currently used local network is: 192.168.1.0/24

and slips sees a forged packet going from 192.168.1.2 to 10.0.0.1, it will alert

Slips detects the current local network by using the local network of the private ips specified in `client_ips` parameter in `slips.conf`

If no IPs are specified, slips uses the local network of the first private source ip found in the traffic.

This threat level of this detection is low if the source ip is the one outside of local network because it's unlikely. and high if the destination ip is the one outside of local network.

6.28 High entropy DNS TXT answers

Slips check every DNS answer with TXT record for high entropy strings. Encoded or encrypted strings with entropy higher than or equal 5 will then be detected using shannon entropy and alerted by slips.

the entropy threshold can be changed in slips.conf by changing the value of `entropy_threshold`

6.29 Devices changing IPs

Slips stores the MAC of each new IP it sees in conn.log.

Then for every source address in conn.log, slips checks if the MAC of it was used by another IP.

If so, it alerts "Device changing IPs".

6.30 GRE tunnels

Slips uses zeek tunnel.log to alert on GRE tunnels when found. evidence of this type are just informational.

6.31 Invalid DNS resolutions

Some DNS resolvers answer the DNS query to adservers with 0.0.0.0 or 127.0.0.1 as the ip of the domain to block the domain. Slips detects this and sets an informational evidence.

CHAPTER

SEVEN

FEATURES

This Section will contain a list of all features and detections listed in the flowalerts section and the detection modules section and a brief description of how slips works.

7.1 Flow Alerts Module

The module of flow alerts has several behavioral techniques to detect attacks by analyzing the content of each flow alone.

The detection techniques are:

- Long connections
- Successful SSH connections
- Connections without DNS resolution
- DNS resolutions to IPs that were never used
- Connections to unknown ports
- Data exfiltration
- Malicious JA3 hashes
- Connections to port 0
- Multiple reconnection attempts
- Alerts from Zeek: Self-signed certs, invalid certs, port-scans and address scans, and password guessing
- DGA
- Connection to multiple ports
- Malicious SSL certificates
- Young domains
- Bad SMTP logins
- SMTP login bruteforce
- DNS ARPA Scans
- Multiple SSH versions
- Incompatible CN
- Weird HTTP methods

- Non-SSL connections on port 443
- Non-HTTP connections on port 80
- Connection to private IPs
- Connection to private IPs outside the current local network
- High entropy DNS TXT answers
- Devices changeing IPs
- SSH version changing

The details of each detection follows.

7.1.1 Long Connections

Detect connections that are long, except the multicast connections.

By defualt, A connection in considered long if it exceeds 1500 seconds (25 Minutes).

This threshold can be changed `config/slips.conf` by changing the value of `long_connection_threshold`

7.1.2 Incompatible CN

Zeek logs each Certificate CN in ssl.log

When slips enccounters a cn that claims to belong to any of Slips supported orgs (Google, Microsoft, Apple or Twitter) Slips checks if the destination address or the destination server name belongs to these org.

If not, slips generates an alert.

7.1.3 High entropy DNS TXT answers

Slips check every DNS answer with TXT record for high entropy strings. Encoded or encrypted strings with entropy higher than or equal 5 will then be detected using shannon entropy and alerted by slips.

the entropy threshold can be changed in slips.conf by changing the value of `entropy_threshold`

7.1.4 Devices changing IPs

Slips stores the MAC of each new IP it sees in conn.log.

Then for every source address in conn.log, slips checks if the MAC of it was used by another IP.

If so, it alerts "Device changing IPs".

7.1.5 SMTP login bruteforce

Slips alerts when 3+ invalid SMTP login attempts occurs within 10s

7.1.6 Connection to private IPs

Slips detects when a private IP is connected to another private IP with threat level info.

But it skips this alert when it's a DNS connection on port 53 UDP to the gateway

7.1.7 Connection to private IPs outside the current local network

Slips detects the currently used local network and alerts if it find a connection to/from a private IP that doesn't belong to it.

For example if the currently used local network is: 192.168.1.0/24

and slips sees a forged packet going from 192.168.1.2 to 10.0.0.1, it will alert

7.1.8 Weird HTTP methods

Slips uses zeek's weird.log where zeek logs weird HTTP methods seen in http.log

When there's a weird HTTP method, slips detects it as well.

7.1.9 Non-SSL connections on port 443

Slips detects established connections on port 443 that are not using HTTP using zeek's conn.log flows

if slips finds a flow using destination port 443 and the 'service' field in conn.log isn't set to 'ssl', it alerts

7.2 Non-HTTP connections on port 80.

Slips detects established connections on port 80 that are not using SSL using zeek's conn.log flows

if slips finds a flow using destination port 80 and the 'service' field in conn.log isn't set to 'http', it alerts

7.2.1 Connections without DNS resolution

This will detect connections done without a previous DNS resolution. The idea is that a connection without a DNS resolution is slightly suspicious.

If Slips runs by capturing packets directly from a network device (as opposed to, for example, a PCAP file), this detection will ignore all connections that happen in the first 3 minute of operation of Slips. This is because most times Slips is started when the computer is already running, and many DNS connections were already done. So waiting 3 minutes decreases the amount of False Positives.

This detection will ignore certain IP addresses for which a connection without DNS is ok. The exceptions are:

- Private IPs
- Localhost IPs (127.0.0.0/8)

- Reserved IPs (including the super broadcast 255.255.255.255)
- IPv6 local-link IPs
- Multicast IPs
- Broadcast IPs only if they are private
- Well known organizations

DNS resolutions of well known orgs might be done using DoH, in this case, slips doesn't know about the DNS resolution because the resolved domain won't be in dns.log so we simply ignore alerts of this time about well known org such as (facebook, apple, google, twitter, and microsoft)

Slips uses it's own lists of organizations info (IPs, IP ranges, domains, and ASNs) stored in `slips_files/organizations_info` to check whether the IP/domain of each flow belong to a known org or not.

Slips doesn't detect 'connection without DNS' when running on an interface except for when it's done by this instance's own IP.

check DoH section of the docs for info on how slips detects DoH.

7.2.2 Successful SSH connections

Slips detects successful SSH connections using 2 ways

1. Using Zeek. Zeek logs successful SSH connection to ssh.log by default
2. if all bytes sent in a SSH connection is more than 4290 bytes

7.2.3 DNS resolutions without a connection

This will detect DNS resolutions for which no further connection was done. A resolution without a usage is slightly suspicious.

The domains that are excepted are:

- All reverse DNS resolutions using the in-addr.arpa domain.
- All .local domains
- The wild card domain *
- Subdomains of cymru.com, since it is used by the ipwhois library to get the ASN of an IP and its range.
- Ignore WPAD domain from Windows
- Ignore domains without a TLD such as the Chrome test domains.

7.2.4 Connection to unknown ports

Slips has a list of known ports located in `slips_files/ports_info/ports_used_by_specific_orgs.csv`

It also has a list of ports that belong to a specific organization in `slips_files/ports_info/ports_used_by_specific_orgs.csv`

For example, even though 5223/TCP isn't a well known port, Apple uses it in Apple Push Notification Service (APNS).

Any port that isn't in the above 2 files is considered unknown to Slips.

Example of Spyware that uses custom ports are hermit using ports 58442/TCP and 8442/TCP.

7.2.5 Data exfiltration

Slips generate a ‘possible data exfiltration alerts when the number of uploaded files to any IP exceeds 700 MBs.

The number of MBs can be modified by editting the value of `data_exfiltration_threshold` in `config/slips.conf`

7.2.6 Malicious JA3 and JA3s hashes

Slips uses JA3 hashes to detect C&C servers (JA3s) and infected clients (JA3)

Slips is shipped with it’s own zeek scripts that add JA3 and JA3s fingerprints to the SSL log files generated by zeek.

Slips supports JA3 feeds in addition to having more than 40 different threat intelligence feeds. The JA3 feeds contain JA3 fingerprints that are identified as malicious. The JA3 threat intelligence feed used by Slips now is Abuse.ch JA3 feed. And you can add other JA3 TI feeds in `ja3_feeds` in `config/slips.conf`.

7.2.7 Connections to port 0

There has been a significant rise in the number of attacks listed as Port 0. Last year, these equated to 10% of all attacks, but now it’s up to almost 25%.

Slips detects any connection to port 0 using any protocol other than ‘IGMP’ and ‘ICMP’ as malicious.

7.2.8 Multiple reconnection attempts

Multiple reconnection attempts in Slips are 5 or more not established flows (reconnections) to the same destination IP.

7.2.9 Zeek alerts

By default, Slips depends on Zeek for detecting different behaviours, for example Self-signed certs, invalid certs, port-scans and address scans, and password guessing.

Some scans are also detected by Slips independently of Zeek, like ICMP sweeps and vertical/horizontal portscans. Check section for more info #todo

7.2.10 DGA

When the DNS server fails to resolve a domain, it responds back with NXDOMAIN code.

To detect DGA, Slips will count the amount of NXDOMAINs met in the DNS traffic of each source IP.

Then we alert when there is 10 or more NXDOMAINs.

Every 10,15,20 ..etc slips generates an evidence.

7.2.11 Connection to multiple ports

When Slips encounters a connection to or from a specific IP and a specific port, it scans previous connections looking for connection to/from that same IP using a different port.

It alerts when finding two or more connections to the same IP.

7.2.12 Malicious SSL certificates

Slips uses SSL certificates sha1 hashes to detect C&C servers.

Slips supports SSL feeds and is shipped with Abuse.ch feed of malicious SSL hashes by default. And you can add other SSL feeds in `ssl_feeds` in `config/slips.conf`.

7.2.13 Young Domains

Slips uses whois python library to get the creation date of every domain met in the dns flows.

If a domain's age is less than 60 days, slips sets an alert.

Not all domains are supported, here's the list of supported TLDs.

```
['.ac_uk', '.am', '.amsterdam', '.ar', '.at', '.au',
'.bank', '.be', '.biz', '.br', '.by', '.ca', '.cc',
'.cl', '.club', '.cn', '.co', '.co_il', '.co_jp', '.com',
'.com_au', '.com_tr', '.cr', '.cz', '.de', '.download', '.edu',
'.education', '.eu', '.fi', '.fm', '.fr', '.frl', '.game', '.global_',
'.hk', '.id_', '.ie', '.im', '.in_', '.info', '.ink', '.io',
'.ir', '.is_', '.it', '.jp', '.kr', '.kz', '.link', '.lt', '.lv',
'.me', '.mobi', '.mu', '.mx', '.name', '.net', '.ninja',
'.nl', '.nu', '.nyc', '.nz', '.online', '.org', '.pe',
'.pharmacy', '.pl', '.press', '.pro', '.pt', '.pub', '.pw',
'.rest', '.ru', '.ru_rf', '.rw', '.sale', '.se', '.security',
'.sh', '.site', '.space', '.store', '.tech', '.tel', '.theatre',
'.tickets', '.trade', '.tv', '.ua', '.uk', '.us', '.uz', '.video',
'.website', '.wiki', '.work', '.xyz', '.za']
```

7.2.14 Bad SMTP logins

Slips uses zeek to detect SMTP connections, When zeek detects a bad smtp login, it logs it to smtp.log, then slips reads this file and sets an evidence.

7.2.15 SMTP bruteforce

Slips detects a SMTP bruteforce when 3 or more bad SMTP logins happen within 10 seconds.

With every generated evidence, Slips gathers as much info about the malicious IP and prints it with the alert.

So instead of having an alerts saying:

```
Detected SSL certificate validation failed with (certificate has expired) Destination␣
↪IP: 216.58.201.70.
```

Slips gathers AS, hostname, SNI, rDNS and any available data about this IP and you get an alert saying:

```
Detected SSL certificate validation failed with (certificate has expired) Destination IP:
216.58.201.70. AS: GOOGLE, US, SNI: 2542116.fls.doubleclick.net, rDNS: prg03s01-in-f70.
↪1e100.net
```

7.2.16 DNS ARPA Scans

Whenever slips sees a new domain in dns.log, if the domain ends with '.in-addr.arpa' slips keeps trach of this domain and the source IP that made the DNS request.

Then, if the source IP is seen doing 10 or more ARPA queries within 2 seconds, slips generates an ARPA scan detection.

7.2.17 SSH version changing

Zeek logs the used software and software versions in software.log, so slips knows from this file the software used by different IPs, like whether it's an SSH::CLIENT, an HTTP::BROWSER, or an HTTP::SERVER

When slips detects an SSH client or an SSH server, it stores it with the IP and the SSH versions used in the database

Then whenever slips sees the same IP using another SSH version, it compares the stored SSH versions with the current SSH versions

If they are different, slips generates an alert

7.3 Detection modules

Slips is a behavioral-based IPS that uses machine learning to detect malicious behaviors in the network traffic. It is a modular software that can be extended. When Slips is run, it spawns several child processes to manage the I/O, to profile attackers and to run the detection modules.

Here we describe what detection modules are run on the traffic to detect malicious behaviour.

Modules are Python-based files that allow any developer to extend the functionality of Slips. They process and analyze data, perform additional detections and store data in Redis for other modules to consume. Currently, Slips has the following modules:

7.3.1 Virustotal Module

This module is used to lookup IPs, domains, and URLs on virustotal

To use it you need to add your virustotal API key in `config/vt_api_key`

7.3.2 RiskIQ Module

This module is used to get different information (passive DNS, IoCs, etc.) from RiskIQ To use this module your RiskIQ email and API key should be stored in `config/RiskIQ_credentials`

the format of this file should be the following:

```
example@domain.com
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

The hash should be your 64 character API Key.

The path of the file can be modified by changing the `RiskIQ_credentials_path` parameter in `config/slips.conf`

7.3.3 Leak Detection Module

This module on runs on pcaps, it uses YARA rules to detect leaks.

You can add your own YARA rule in `modules/leak_detector/yara_rules/rules` and it will be automatically compiled and stored in `modules/leak_detector/yara_rules/compiled` and matched against every pcap.

7.3.4 Blocking Module

To enable blocking in slips, start slips with the `-p` flag.

This feature is only supported in linux using iptables natively and using docker.

7.3.5 Exporting Alerts Module

Slips supports exporting alerts to other systems using different modules (ExportingAlerts, CESNET sharing etc.)

For now the supported systems are:

- Slack
- TAXII Servers (STIX format)
- Warden servers
- IDEA JSON format
- Logstash

Refer to the exporting section of the docs for detailed instructions on how to export.

7.3.6 Flowalerts Module

This module is responsible for detecting malicious behaviours in your traffic.

Refer to the Flowalerts section of the docs for detailed explanation of what Slips detects and how it detects.

7.3.7 Disabled alerts

All Slips detections are turned on by default, You can configure which alerts you want to enable/disable in `config/slips.conf`

Slips support disabling unwanted alerts, simply add the detection you want to disable in the `disabled_detections` list and slips will not generate any alerts of this type.

for example:

```
disabled_detections = [MaliciousJA3, DataExfiltration, SelfSignedCertificate]
```

Supported detections are:

ARPScan, ARP-outside-localnet, UnsolicitedARP, MITM-ARP-attack, SSHSuccessful, LongConnection, MultipleReconnectionAttempts, ConnectionToMultiplePorts, InvalidCertificate, UnknownPort, Port0Connection, ConnectionWithoutDNS, DNSWithoutConnection, MaliciousJA3, DataExfiltration, SelfSignedCertificate, PortScanType1, PortScanType2, Password_Guessing, MaliciousFlow, SuspiciousUserAgent, multiple_google_connections, NETWORK_gps_location_leaked, Command-and-Control-channels-detection, ThreatIntelligenceBlacklistDomain, ThreatIntelligenceBlacklistIP, MaliciousDownloadedFile, DGA, MaliciousSSLCert, YoungDomain, MultipleSSHVersions DNS-ARPA-Scan, SMTPLoginBruteforce, BadSMTPLogin, IncompatibleUserAgent, ICMP-Timestamp-Scan, ICMP-AddressScan, ICMP-AddressMaskScan

7.3.8 Threat Intelligence Module

Slips has a complex system to deal with Threat Intelligence feeds.

Slips supports different kinds of IoCs from TI feeds (IPs, IP ranges, domains, JA3 hashes, SSL hashes)

File hashes and URLs aren't supported.

Matching of IPs

Slips gets every IP it can find in the network and tries to see if it is in any blacklist.

If a match is found, it generates an evidence, if no exact match is found, it searches the Blacklisted ranges taken from different TI feeds

Matching of Domains

Slips gets every domain that can find in the network and tries to see if it is in any blacklist. The domains are currently taken from:

- DNS requests
- DNS responses
- HTTP host names
- TLS SNI

Once a domain is found, it is verified against the downloaded list of domains from the blacklists defined in `ti_files` in the configuration file `config/slips.conf`. If an exact match is found, then an evidence is generated.

If an exact match is not found, then Slips verifies if the verified domain is a subdomain of any domain in the blacklist.

For example, if the domain in the traffic is *here.testing.com*, Slips first checks if the exact domain *here.testing.com* is in any blacklist, and if there is no match, it checks if the domain *testing.com* is in any blacklists too.

Matching of JA3 Hashes

Every time Slips encounters an TLS flow, it compares each JA3 and JA3s with the feeds of malicious JA3 and alerts when there's a match. Slips is shipped with the Abuse.ch JA3 feed by default You can add your own SSL feed by appending to the file defined by the `ja3_feeds` key in `config/slips.conf`, which is by default `config/JA3_feeds.csv`

Matching of SSL SHA1 Hashes

Every time Slips encounters an SSL flow, it tries to get the certificate hash from zeek ssl.log, then it compares the hash with our list of blacklisted SSL certificates

Slips is shipped with the Abuse.ch SSL feed by default,

You can add your own SSL feed by appending to the file defined by the `ssl_feeds` key in `config/slips.conf`, which is by default `config/SSL_feeds.csv`

Local Threat Intelligence files

Slips has a local file for adding IoCs of your own, it's located in `config/local_data_files/own_malicious_iocs.csv` by default, this path can be changed by changing `download_path_for_local_threat_intelligence` in `config/slips.conf`.

The format of the file is "IP address","Threat level", "Description"

Threat level available options: info, low, medium, high, critical

Refer to the architecture section of the docs for detailed explanation of Slips threat levels.

Example:

```
"23.253.126.58","high","Simda CC"
"bncv00.no-ip.info", "critical", "Variant.Zusy"
```

Local JA3 hashes

Slips has a local file for adding JA3 hashes of your own, it's located in `config/local_data_files/own_malicious_JA3.csv` by default.

The format of the file is "JA3 hash", "Threat level", "Description"

Threat level available options: info, low, medium, high, critical

Refer to the architecture section of the docs for detailed explanation of Slips threat levels.

Example:

```
"e7d705a3286e19ea42f587b344ee6865","medium","Standard tor client"
"6734f37431670b3ab4292b8f60f29984", "high", "Trickbot Malwar"
```

Adding your own remote feed

We update the remote ones regularly. The list of remote threat intelligence files is set in the variables `ti_files` variable in config/slips.conf. You can add your own remote threat intelligence feeds in this variable. Supported extensions are: .txt, .csv, .netset, ipsum feeds, or .intel.

Each URL should be added with a threat_level and a tag, the format is (url,threat_level,tag)

tag is which category is this feed e.g. phishing, adtrackers, etc..

Threat level available options: info, low, medium, high, critical

Refer to the architecture section of the docs for detailed explanation of Slips threat levels.

TI files commented using # may be processed as they're still in our database.

Use `;` for commenting TI files in `config/slips.conf` instead of `#`.

Commented TI files (lines starting with ;) will be completely removed from our database.

The remote files are downloaded to the path set in the `download_path_for_local_threat_intelligence`. By default, the files are stored in the Slips directory `modules/ThreatIntelligence1/remote_data_files/`

Commenting a remote TI feed

If you have a remote file link that you wish to comment and remove from the database you can do so by adding ';' to the line that contains the feed link in `config/slips.conf`, don't use the '#' for example to comment the `bruteforcelogin` feed you should do the following:

```
;https://lists.blocklist.de/lists/bruteforcelogin.txt,medium,['honeypot']
```

instead of:

```
#https://lists.blocklist.de/lists/bruteforcelogin.txt,medium,['honeypot']
```

7.3.9 Update Manager Module

To make sure Slips is up to date with the most recent IoCs in all feeds, all feeds are loaded, parsed and updated periodically and automatically by Slips every 24 hours, which requires no user interaction.

The 24 hours interval can be changed by changing the `TI_files_update_period` key in `config/slips.conf`

Update manager is responsible for updating all remote TI files (including SSL and JA3 etc.)

By default, local slips files (organization_info, ports_info, etc.) are cached to avoid loading and parsing them everytime we start slips. However, they are updated automatically by the update manager if they were changed on disk.

7.3.10 IP Info Module

The IP info module has several ways of getting information about an IP address, it includes:

- ASN
- Country by Geolocation
- Given a MAC, its Vendor
- Reverse DNS

ASN

Slips is shipped with an offline database (GeoLite2) in `databases/GeoLite2-ASN.mmdb` to search for ASNs, if the ASN of a given IP is not in the GeoLite2 database, we try to get the ASN online using the online database using the `ipwhois` library. However, to reduce the amount of requests, we retrieve the range of the IP and we cache the whole range. To search and cache the whole range of an IP, the module uses the ipwhois library. The ipwhois library gets the range of this IP by making a connection to the server `cymru.com` using a TXT DNS query. The DNS server is the one set up in the operating system. For example to get the ASN of the IP 13.32.98.150, you will see a DNS connection asking for the TXT record of the domain `150.98.32.13.origin.asn.cymru.com`.

Country by Geolocation

Slips is shipped with an offline database (GeoLite2) in `databases/GeoLite2-Country.mmdb` to search for Geolocation.

Mac Vendors

Slips is shipped with an offline database `databases/macaddress-db.json` for MAC address vendor mapping.

This database is a combination of 2 different online databases, but the format of them is changed to a format slips understands and to reduce the size of the db.

Slips gets the MAC address of each IP from dhcp.log and arp.log and then searches the offline database using the OUI.

If the vendor isn't found in the offline MAC database, Slips tries to get the MAc using the online database https://www.macvendorlookup.com

The offline database is updated manually and shipped with slips, you can find it in the `databases/` dir.

Slips makes sure it doesn't perform duplicate searches of the same MAC Address either online, or offline.

Reverse DNS

This is obtained by doing a standard in-addr.arpa DNS request.

7.3.11 ARP Module

This module is used to check for ARP attacks in your network traffic.

By default, zeek doesn't generate and log ARP flows, but Slips is shipped with it's own zeek scripts that enable the logging of ARP flows in `arp.log`

The detection techniques are:

- ARP scans
- ARP to a destination IP outside of local network
- Unsolicited ARP
- MITM ARP attack

ARP Scans

Slips considers an IP performing an ARP scan if it sends 5 or more non-gratuitous ARP to different destination addresses in 30 seconds or less.

ARP to a destination IP outside of local network

Slips alerts when an ARP flow is being sent to an IP outside of local network as it's a weird behaviour that shouldn't be happening.

Unsolicited ARP

Unsolicited ARP is used to update the neighbours' ARP caches but can also be used in ARP spoofing, we detect it with threat level 'info', so we don't consider it malicious, we simply notify you about it.

MITM ARP attack

Slips detects when a MAC with IP A, is trying to tell others that now that MAC is also for IP B (ARP cache attack)

7.3.12 CESNET sharing Module

This module is responsibe for importing and exporting alerts from and to warden server

Refer to the exporting section of the docs for detailed instructions on CESNET exporting and the format of the configuration files.

To enable the importing alerts from warden servers, set `receive_alerts` to `yes` in config/slips.conf

Slips imports 100 alerts from warden servers each day, and automatically stores the aleerts in our database

Time to wait before receiving alerts from warden server is 1 day by default, you can change this by chaning the `receive_delay` in `config/slips.conf`

These are the categories we import: ['Availability', 'Abusive.Spam','Attempt.Login', 'Attempt', 'Information', 'Fraud.Scam', 'Information', 'Fraud.Scam']

7.3.13 HTTP Analyzer Module

This module handles the detections of HTTP flows

Available detection are:

- Multiple empty connections
- Suspicious user agents
- Incompatible user agents
- Multiple user agents
- Downloads from pastebin
- Executable downloads

Multiple empty connections

Due to the usage of empty connections to popular site by malware to check for internet connectivity, We consider this type of behaviour suspicious activity that shouldn't happen

We detect empty connection to 'bing.com', 'google.com', 'yandex.com', 'yahoo.com', 'duckduckgo.com' etc.

Suspicious user agents

Slips has a list of suspicious user agents, whenever one of them is found in the traffic, slips generates and evidence.

Our current list of user agents has: ['httpsend', 'chm_msdn', 'pb', 'jndi', 'tesseract']

Incompatible user agents

Slips uses and offline MAC address database to detect the type of device based on the MAC OUI.

First, Slips store the MAC address and vendor of every IP it sees (if available)

Second, When slips encounters a user agent in HTTP traffic it performs an online query to http://useragentstring.com to get more info about this user agent, like the os type, name and browser.

Third, When slips has both information available (MAC vendor and user agent), it compares them to detect incompatibility using a list of keywords for each operating system.

Available keywords for Apple: ('macos', 'ios', 'apple', 'os x', 'mac', 'macintosh', 'darwin')

Available keywords for Microsoft: ('microsoft', 'windows', 'nt')

Available keywords for Android: ('android', 'google')

Multiple user agents

Slips stores the MAC address and vendor of every IP it sees (if available) in the redis database. Then, when an IP iss seen using a different user agent than the one stored in the database, it tries to extract os info from the user agent string, either by performing an online query to http://useragentstring.com or by using zeek.

If an IP is detected using different user agents that refer to different operating systems, an alert of type 'Multiple user agents' is made

for example, if an IP is detected using a macOS user agent then an android user agent, slips detects this with 'low' threat level

Pastebin downloads

Some malware use pastebin as the host of their malicious payloads.

Slips detects downloads of files from pastebin with size >= 700 bytes.

This value can be customized in slips.conf by changing `pastebin_download_threshold`

When found, slips alerts pastebin download with threat level low because not all downloads from pastebin are malicious.

Executable downloads

Slips generates an evidence everytime there's an executable download from an HTTP website.

7.3.14 Leak Detector Module

This module work only when slips is given a PCAP

The leak detector module uses YARA rules to detect leaks in PCAPs

Module requirements

In order for this module to run you need:

You can install YARA by running

`sudo apt install yara`

You can install tshark by running

`sudo apt install wireshark`

How it works

This module works by

1. Compiling the YARA rules in the `modules/leak_detector/yara_rules/rules/` directory
2. Saving the compiled rules in `modules/leak_detector/yara_rules/compiled/`
3. Running the compiled rules on the given PCAP
4. Once we find a match, we get the packet containing this match and set evidence.

Extending

You can extend the module be adding more YARA rules in `modules/leak_detector/yara_rules/rules/`.

The rules will be automatically detected, compiled and run on the given PCAP.

If you want to contribute, improve existing Slips detection modules or implement your own detection modules, see section :doc:`Contributing <contributing>`.

7.3.15 Network service discovery Module

This module is responsibe for detecting scans such as:

- Vertical port scans
- Horizontal port scans
- PING sweeps
- DHCP scans

Vertical port scans

Slips checks both TCP and UDP connections for port scans.

Slips considers an IP performing a vertical port scan if it scans 6 or more different destination ports

We detect a scan every threshold. So we detect when there is 6, 9, 12, etc. destination ports per destination IP.

Horizontal port scans

Slips checks both TCP and UDP connections for horizontal port scans.

Slips considers an IP performing a horizontal port scan if it contacted more than 3 destination IPs on a specific port with not established connections.

We detect a scan every threshold. So we detect when there is 6, 9, 12, etc. destination destination IPs.

PING Sweeps

PING sweeps or ICMP sweeps is used to find out which hosts are alive in a network or large number of IP addresses using PING/ICMP.

We detect a scan every threshold. So we generate an evidence when there is 5,10,15, .. etc. ICMP established connections to different IPs.

We detect 3 types of ICMP scans: ICMP-Timestamp-Scan, ICMP-AddressScan, and ICMP-AddressMaskScan

Slips does this detection using Slips' own zeek script located in zeek-scripts/icmps-scans.zeek for zeek and pcap files and using the portscan module for binetflow files.

7.4 Connections Made By Slips

Slips uses online databases to query information about many different things, for example (user agents, mac vendors etc.)

The list below contains all connections made by Slips

useragentstring.com -> For getting user agent info if no info was found in Zeek macvendorlookup.com -> For getting MAC vendor info if no info was found in the local maxmind db ip-api.com/json/ -> For getting ASN info about IPs if no info was found in our Redis DB ipinfo.io/json -> For getting your public IP virustotal.com -> For getting scores about downloaded files, domains, IPs and URLs cymru.com -> For getting the range of a specific IP to cache the ASN of this range. TXT DNS queries are made to this domain.

By default, slips whitelists alerts from or to any of the above domains, witch means that if an alert was detected to one of the above alerts, slips does not detect it assuming it's a false positive and the connection was made internally by slips.

You can change this behaviour by updating `whitelist.conf`.

7.5 Ensembling

Ensembling in Slips is done by the Evidence Process.

Every time the evidence process gets a new evidence from the detection modules, it retrieves all the past evidence by this the source IP, in the current timewindow from the datbase.

Then, Slips uses the following equation to get the score of each evidence

threat_level = threat_level * confidence

Slips accumulates the threat level of all evidenc, then, it checks if the accumulated threat level reached a certain threshold or not.

If the accumulated threat level reached the threshold specified in `evidence_detection_threshold`, Slips generates and alert. If not, slips waits for the next evidence, accumulates threat levels, and checks again until the threshold is reached.

7.6 Controlling Slips Sensitivity

The threshold that controls Slips sensitivity is determined by the `evidence_detection_threshold` key in `config/slips.conf`, by default it is set to `3.46`.

This threshold is the minimum accumulated threat level per time window needed to generate an alert.

The default threshold of 3.46 gives you balanced detections with the optimal false positive rate and accuracy.

The Optimal range is from 3.1 to 3.89. The higher the value in this range, the less false positives and the less accuracy you get.

Here are more options

- **0.2**: Use this threshold If you want Slips to be super sensitive with higher FPR, using this means you are less likely to miss a detection but more likely to get false positives.
- **6.3**: Use this threshold If you want Slips to be insensitive. meaning Slips will need so much evidence to trigger an alert. May lead to false negatives.
- **3.1**: The start of the optimal range, has more false positives but more accurate.
- **3.86**: The end of the optimal range, has less false positives but less accurate.

7.7 Zeek Scripts

Slips is shipped with it's own custom zeek scripts to be able to extend zeek functionality and customize the detections

7.7.1 Detect DoH

In the `detect_DoH.zeek` script, slips has it's own list of ips that belong to dns/doh servers,

When slips encouters a connection to any IP of that list on port 443/tcp, it assumes it's a DoH connetion,

and times out the connection after 1h so that the connection won't take too long to appear in slips.

7.7.2 Detect ICMP Scans

In the `zeek-scripts/icmps-scans.zeek` script, we check the type of ICMP in every ICMP packet seen in the network,

and we detect 3 types of ICMP scans: ICMP-Timestamp-Scan, ICMP-AddressScan, and ICMP-AddressMaskScan based on the icmp type

We detect a scan every threshold. So we generate an evidence when there is 5,10,15, .. etc. ICMP established connections to different IPs.

7.7.3 CPU Profiling

Slips is shipped with its own tool for CPU Profiling, it can be found it `slips_files/common/cpu_profiler.py`

CPU Profiling supports 2 modes: live and development mode

Live mode:

The main purpose of this mode it to show live CPU stats in the web interface. "live" mode publishes updates during the runtime of the program to the redis channel 'cpu_profile' so that the web interface can use them

Development mode:

Setting the mode to "dev" outputs a JSON file of the CPU usage at the end of the program run. It is recommended to only use dev mode for static file inputs (pcaps, suricata files, binetflows, etc.) instead of interface and growing zeek dirs, because longer runs result in profiling data loss and not everything will get recorded. The JSON file created in this mode is placed in the output dir of the current run and can be viewed by running the following command

`vizviewer results.json`

then going to http://127.0.0.1:9001/ in your browser for seeing the visualizations of the CPU usage

Options to enable cpu profiling can be found under the [Profiling] section of the `slips.conf` file. `cpu_profiler_enable` set to "yes" enables cpu profiling, or "no" to disable it. `cpu_profiler_mode` can be set to "live" or "dev". Setting to `cpu_profiler_multiprocess` can be set to "yes" or "no" and only affects the dev mode profiling. If set to "yes" then all processes will be profiled. If set to "no" then only the main process (slips.py) will be profiled. `cpu_profiler_output_limit` is set to an integer value and only affects the live mode profiling. This option sets the limit on the number of processes output for live mode profiling updates. `cpu_profiler_sampling_interval` is set to an integer value and only affects the live mode profiling. This option sets the duration in seconds of live mode sampling intervals. It is recommended to set this option greater than 10 seconds otherwise there won't be much useful information captured during sampling.

7.7.4 Memory Profiling

Memory profiling can be found in `slips_files/common/memory_profiler.py`

Just like CPU profiling, it also has supports live and development mode. Set `memory_profiler_enable` to `yes` to enable this feature. Set `memory_profiler_mode` to `live` to use live mode or `dev` to use development mode profiling.

Live Mode

This mode shows memory usage stats during the runtime of the program. `memory_profiler_multiprocess` controls whether live mode tracks all processes or only the main process. If set to no, the program will wait for you to connect from a different terminal using the command `memray live <port_number>`, where port_number is 5000 by default. After connection, the program will continue with its run and the terminal that is connected will receive a feed of the memory statistics. If set to yes, the redis channel "memory_profile" can be used to set pid of the process to be tracked. Only a single process can be tracked at a time. The interface is cumbersome to use from the command line so multiprocess live profiling is intended to be used primarily from the web interface.

Development Mode

When enabled, the profiler will output the profile data into the output directory. The data will be in the `memoryprofile` directory of the output directory of the run. Each process during the run of the program will have an associated binary file. Each of the generated binaries will automatically be converted to viewable html files, with each process converted to a flamegraph and table format. All generated files will be denoted by their PID.

If you want to contribute: improve existing Slips detection modules or implement your own detection modules, see section :doc:`Contributing <contributing>`.

CHAPTER

EIGHT

TRAINING

Slips has one machine learning module that can be retrained by users. This is done by puttin slips in training mode so you can re-train the machine learning models with your own traffic. By default Slips includes an already trained model with our data, but it is sometimes necessary to adapt it to your own circumstances.

Until Slips 0.7.3, there is only one module for now that can do this, the one called 'flowmldetection'. This module analyzes flows one by one, as formatted similarly as in a conn.log Zeek file. This module is enabled by default in testing mode. This module uses by default the SGDClassifier with a linear support vector machine (SVM). The decision to use SVM was done because is one of the few algorithms that can be used for online learning and that can extend a current model with new data.

To re-train this machine learning algorithm, you need to do the following:

1- Edit the config/slips.conf file to put Slips in train mode. Search the word **train** in the section **[flowmldetection]** and uncomment the **mode = train** and comment **mode = test**. It should look like

```
[flowmldetection]
# The mode 'train' should be used to tell the flowmldetection module that the flows␣
↪received are all for training.
# A label should be provided in the [Parameters] section
mode = train

# The mode 'test' should be used after training the models, to test in unknown data.
# You should have trained at least once with 'Normal' data and once with 'Malicious' data␣
↪in order for the test to work.
#mode = test
```

2- Establish the general label for all the traffic that you want to re-train with. For now we only support 1 label per file. Search in the [parameters] section and choose the type of traffic you will send to Slips.

```
# Set the label for all the flows that are being read. For now only normal and malware␣
↪directly. No option for setting labels with a filter
label = normal
#label = malicious
#label = unknown
```

After this edits, just run Slips as usual with any type of input, for example with a Zeek folder.

```
./slips.py -c config/slips.conf -f ~/my-computer-normal/
```

Or with a pcap file.

```
./slips.py -c config/slips.conf -f ~/my-computer-normal2.pcap
```

3- If you have also malicious traffic, first change the label to malicious in config/slips.conf

```
# Set the label for all the flows that are being read. For now only normal and malware␣
↪directly. No option for setting labels with a filter
#label = normal
label = malicious
#label = unknown

./slips.py -c config/slips.conf -f ~/my-computer-normal2.pcap
```

After this edits, just run Slips as usual with any type of input, for example another pcap

```
./slips.py -c config/slips.conf -f ~/malware1.pcap
```

You can also run slips in an interface and train it directly with your data

```
./slips.py -c config/slips.conf -i eth0
```

4- Finally to use the model, put back the **test** mode in the configuration config/slips.conf

```
[flowmldetection]
# The mode 'train' should be used to tell the flowmldetection module that the flows␣
↪received are all for training.
# A label should be provided in the [Parameters] section
#mode = train

# The mode 'test' should be used after training the models, to test in unknown data.
# You should have trained at least once with 'Normal' data and once with 'Malicious' data␣
↪in order for the test to work.
mode = test
```

5- Use slips normally in files or interfaces

```
./slips.py -c config/slips.conf -i eth0
```

CHAPTER

NINE

EXPORTING

Slips supports exporting alerts to other systems using different modules (ExportingAlerts, CESNET sharing etc.)

For now the supported systems are:

- Slack
- TAXII Servers (STIX format)
- Warden servers
- IDEA JSON format
- Logstash
- TSV and json of labeled flows

9.1 Slack

Slips uses the WebHook method to send data to Slack, more info here.

To export into a slack channel you need to:

1. Create a new application in your slack, see `https://api.slack.com/apps/` Remember that applications are seen per user, so other users in your Slack will not see this application probably.
2. Activate Incoming Webhooks while creating your app.
3. Create an Incoming Webhook for the channel you want to send the messages too.
4. Go to Slack and copy the channel ID for this channel. You can do this by going to the channel, then clicking on the channel's name. The ID is in the bottom of the pop-up window.
5. You need to give your app the correct scope. Slips only needs write access to one channel. Do: 5.1 Go to your app in Slack `https://api.slack.com/apps` 5.2 In the navigation menu, choose the OAuth & Permissions feature. 5.3 Scroll down to the Scopes section, and pick channels:read and chat:write from the drop down menu. 5.4 Scroll back to the top of this page and look for the button that says Install App to Workspace (or Reinstall App if you've done this before). Click it.
6. In this same 'OAuth & Permissions' page, copy the 'Bot User OAuth Token'. It should look something like 'xoxb-nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn' with a lot of letters.
7. Put the bot OAuth token to the file: `config/slack_bot_token_secret`
8. You need to add the new app to the channel in Slack. You do this by clicking on the bot's name (is in the messae when you add an integration in the channel), and click 'Add this app to a channel'.
9. Alternatively you can add the bot to the channel by going to the channel and doing `/invite @bots_name`.

10. Edit the config/slips.conf file, put `slack` in the export_to variable, and add the channel's name to which you want to send.

    [exporting_alerts] export_to = [slack] slack_channel_name = SlipsAlertsChannel

9.2 STIX

If you want to export alerts to your TAXII server using STIX format, change `export_to` variable to export to STIX, and Slips will automatically generate a `STIX_data.json` containing all alerts it detects.

```
[ExportingAlerts]
export_to = [stix]
```

You can add your TAXII server details in the following variables:

`TAXII_server`: link to your TAXII server

`port`: port to be used

`use_https`: use https or not.

`discovery_path` and `inbox_path` should contain URIs not full urls. For example:

```
discovery_path = /services/discovery-a
inbox_path = /services/inbox-a
```

`collection_name`: the collection on the server you want to push your STIX data to.

`push_delay`: the time to wait before pushing STIX data to server (in seconds). It is used when slips is running non-stop (e.g with -i)

`taxii_username`: TAXII server user credentials

`taxii_password`: TAXII server user password

`jwt_auth_path`: auth path if JWT based authentication is used. It's usually /management/auth. this is what we use to get a token.

if your TAXII server is a remote server, you can set the `port` to 443 or 80.

If running on a file, Slips will export to server after analysis is done. If running on an interface, Slips will export to server every push_delay seconds. by default it's 1h.

9.3 JSON format

By default Slips logs all alerts to `output/alerts.json` in CESNET's IDEA0 format which is also a JSON format.

9.4 CESNET Sharing

Slips supports exporting alerts to warden servers, as well as importing alerts.

To enable the exporting, set `receive_alerts` to `yes` in config/slips.conf

The default configuration file path is specified in the `configuration_file` variable in `config/slips.conf`

The default path is `config/warden.conf`

The format of `warden.conf` should be the following:

```
{ "url": "https://example.com/warden3",
 "certfile": "cert.pem",
 "keyfile": "key.pem",
 "cafile": "/etc/ssl/certs/DigiCert_Assured_ID_Root_CA.pem",
 "timeout": 600,
 "errlog": {"file": "output/warden_logs/warden.err", "level": "debug"},
 "filelog": {"file": "output/warden_logs/warden.log", "level": "warning"},
 "name": "com.example.warden.test" }
```

To get your key and the certificate, you need to run `warden_apply.sh` with you registered client_name and password. Full instructions here

The `name` key is your registered warden node name.

All evidence causing an alert are exported to warden server once an alert is generated. See the difference between alerts and evidence) in Slips architecture section.

You can change how often you get alerts (import) from warden server

By default Slips imports alerts every 1 day, you can change this by changing the `receive_delay` value in `config/slips.conf`

Slips logs all alerts to `output/alerts.json` in CESNET's IDEA0 format by default.

Make sure that the DigiCert_Assured_ID_Root_CA is somewhere accessible by slips. or run slips with root if you want to leave it in `/etc/ssl/certs/`

Refer to the Detection modules section of the docs for detailed instructions on how CESNET importing.

9.5 Logstash

Slips has logstash.conf file that exports our alerts.json to a given output file, you can change the output to your preference (for example: elastic search, stdout, etc.)

9.6 Text logs

By default, the output of Slips is stored in the `output/` directory in two files:

1. alert.json in IDEA0 format
2. alerts.log human readable text format

9.7 TSV and json of labeled flows

Slips supports exporting all the labeled flows and altflows stored in the sqlite database the sqlite database can be exported to json or tsv format.

Each labeled flow has an AID fingerprint, which is used to identify the flow based on the ts, source and destination address, source and destination port and protocol.

this can be done by setting the `export_labeled_flows` parameter to `yes` in slips.conf and changing the `export_format` parameter to your desired format. for now, the `export_format` parameter supports tsv or json formats only.

the exported flows are stored in a file called `labeled_flows.json` or `labeled_flows.tsv` in the output directory.

CHAPTER

TEN

P2P

The P2P module makes Slips be a peer in a peer to peer network of computers in the local network. The peers are only in the local network and they communicate using multicast packets. The P2P module is a highly complex system of data sharing, reports on malicious computers, asking about IoC to the peers and a complex trust model that is designed to resiste adversarial peers in the network. Adversarial peers are malicious peers that lie about the data being shared (like saying that a computer is maliciuos when is not, or that an attacker is benign).

This module was designed and partially implemented in a Master Thesis on CTU FEL by Dita Hollmannova. The goal was to enable Slips instances in a local network to share detections and collectively improve blocking decisions. While the thesis succeeded in creating a framework and a trust model, the project is far from stable. The final implementation in Slips was finished by Alya Gomaa.

This readme provides a shallow overview of the code structure, to briefly document the code for future developers. The whole architecture was thoroughly documented in the thesis itself, which can be downloaded from the link above.

The basic structure of the P2P system is (i) an Slips P2P module in python (called Dovecot), and (ii) a P2P communication system done in Golang (called Pigeon).

10.1 Pigeon

Pigeon is written in golang and is developed in an independent repository from Slips, but is included as a submodules of Slips repository. https://github.com/stratosphereips/p2p4slips

Pigeon handles the P2P communication in the network using the libp2p library, and provides a simple interface to the Slips module. A compiled Pigeon binary is included in the module for convenience.

Pigeon uses the JSON format to communicate with the module or with other Pigeons. For details on the communication format, see the thesis.

10.2 Docker direct use

You can use Slips with P2P directly in a special docker image by doing:

```
docker pull stratosphereips/slips_p2p
docker run -it --rm --net=host --cap-add=NET_ADMIN stratosphereips/slips_p2p
```

For the p2p to be able to listen on the network interfaces and receive packets you should use `--cap-add=NET_ADMIN`

10.3 Installation:

1. download and install go:

```
apt install golang
```

or by hand

```
curl https://dl.google.com/go/go1.18.linux-amd64.tar.gz --output go.tar.gz
rm -rf /usr/local/go && tar -C /usr/local -xzf go.tar.gz
export PATH=$PATH:/usr/local/go/bin
```

1. build the pigeon:

- if you installed slips with the submodules using

```
git clone --recurse-submodules --remote-submodules https://github.com/stratosphereips/
↪StratosphereLinuxIPS -j4
```

then you should only build the pigeon using: `cd p2p4slips && go build`

- If you installed Slips without the submodules then you should download and build the pigeon using:

```
git submodule init && git submodule update && cd p2p4slips && go build
```

The p2p binary should now be in `p2p4slips/` dir and slips will be able to find it.

NOTE

If you installed the p2p4slips submodule anywhere other than slips main directory, remember to add it to PATH by using the following commands:

```
echo "export PATH=$PATH:/path/to/StratosphereLinuxIPS/p2p4slips/" >> ~/.bashrc
source ~/.bashrc
```

10.4 Usage in Slips

The P2P module is disabled by default in Slips.

To enable it, change `use_p2p=no` to `use_p2p=yes` in `config/slips.conf`

P2P is only available when running slips in you local network using an interface. (with -i)

You don't have to do anything in particular for the P2P module to work, just enable it and Slips will: 1- Automatically find other peers in the network (and remember them even if they go offline for days)

2- Ask the group of peers (the network) about what they think of some IoC

3- Group the answers and give Slips an aggregated, balanced, normalized view of the network opinion on each IoC

4- Send blame reports to the whole network about attackers

5- Receive blame reports on attackers from the network, balanced by the trust model

6- Keep a trust score on each peer, which varies in time based on the interactions and quality of data shared

10.5 Project sections

The project is built into Slips as a module and uses Redis for communication. Integration with Slips is seamless, and it should be easy to adjust the module for use with other IPSs. The following code is related to Dovecot:

- Slips, the Intrusion Prevention System
- Dovecot module, the module for Slips
- Pigeon, a P2P wrapper written in golang
- Dovecot experiments, a framework for evaluating trust models (optional)

10.6 Dovecot experiments

Experiments are not essential to the module, and the whole project runs just fine without them. They are useful for development of new trust models and modelling behavior of the P2P network.

To use the experiments, clone the https://github.com/stratosphereips/p2p4slips-experiments repository into `modules/p2ptrust/testing/experiments`.

The experiments run independently (outside of Slips) and start all processes that are needed, including relevant parts of Slips. The code needs to be placed inside the module, so that necessary dependencies are accessible. This is not the best design choice, but it was the simplest quick solution.

10.7 How it works:

Slips interacts with other slips peers for the following purposes:

10.7.1 Blaming IPs

If slips finds that an IP is malicious given enough evidence, it blocks it and tells other peers that this IP is malicious and they need to block it. this is called sending a blame report.

10.7.2 Receiving Blames

When slips receives a blame report from the network, which means some other slips instance in th network set an evidence about an IP and is letting other peers know about it.

Slips then generates an evidence about the reported IP and takes the report into consideration when deciding to block the attacker's IP.

10.7.3 Asking the network about an IP

Whenever slips sees a new IP, it asks other peers about it, and waits 3 seconds for them to reply.

The network then replies with a score and confidence for the IP. The higher the score the more malicious this IP is.

Once we get the score of the IP, we store it in the database, and we alert if the score of this IP is more than 0 (threat level=info).

10.7.4 Answering the network's request about an IP

When asked about an ip, slips shares the score of it and the confidence with the requesting peer. the scores are generated by slips and saved in the database.

10.8 Logs

Slips contains a minimal log file for reports received by other peers and peer updates in `output/p2p_reports.log`

For a more detailed p2p logs, for example (peer ping pongs, peer lists, errors, etc.) you can enable p2p.log in slips.conf by setting `create_p2p_logfile` to `yes` and a `p2p.log` will be available in the output dir

Slips rotates the p2p.log every 1 day by default, and keeps the logs of 1 past day only.

10.9 Limitations

For now, slips only supports requests and blames about IPs.

Domains, URLs, or hashes are not supported, but can easily be added in the future.

10.10 TLDR;

Slips only shares scores and confidence (numbers) generated by slips about IPs to the network, no private information is shared.

CHAPTER

ELEVEN

SLIPS IN ACTION

To demonstrate the capabilities of Slips, we will give it real life malware traffic and checking how Slips analyses it.

11.1 Saefko RAT

We provide the analysis of the network traffic of the RAT06-Saefko download here using Slips.

The capture contains different actions done by the RAT controller (e.g. upload a file, get GPS location, monitor files, etc.). For detailed analysis, check Kamila Babayeva's blog Dissecting a RAT. Analysis of the Saefko RAT.

Disclaimer: The used Slips version in this demo is 1.0.2, alerts and evidence generated in this demo may be different than the alerts you may see using the latest version of Slips.

From the analysis we know that:

- The controller IP address: 192.168.131.1 and 2001:718:2:903:f410:3340:d02b:b918
- The victim's IP address: 192.168.131.2 and 2001:718:2:903:b877:48ae:9531:fbfc

First we run slips using the following command:

```
./slips.py -e 1 -f RAT06_Saefko.pcap
```

First, Slips will start by updating all the remote TI feeds added in slips.conf

To make sure Slips is up to date with the most recent IoCs in all feeds, all feeds are loaded, parsed and updated periodically and automatically by Slips every 24 hours by our Update Manager, which requires no user interaction.

Afetr updating, slips modules start and print the PID of every successfully started module.

Then, we see the alert

Alerts are printed by the evidence module, Slips detected IP `2001:718:2:903:b877:48ae:9531:fbfc` as infected due to the above evidence See the difference between alerts and evidence here

Slips splits does detections in timewindows, each time window is 1 hour long by default and contains dozens of features computed for all connections that start in that time window. So if an IP behaves maliciously at 4 PM, it will be marked as infected only during that hour, the next hour if no malicious behaviour occurs, slips will treat the traffic as normal. This explains the start and stop timestamps in the alert `start 2021-04-10T16:44:43.285478+02:00, stop 2021-04-10T17:44:43.285478+0200`. This is the period (timewindow) in which this IP was behaving maliciously.

The difference between infected and normal timewindows is shown better in kalispo, our user interface.

You can start it in another terminal using `./kalipso.sh`

We can see that IP 2001:718:2:903:b877:48ae:9531:fbfc is infected only in timewindow1 as it's marked in red and is behaving normally in timewindow0 as it's colored in green.

We can see all the flows done by this IP in the infected timewindow in kalipso by pressing enter on timewindow1.

At the bottom box in kalipso we can scroll though the evidence and se what slips detected, this is the same evidence printed in Figure 3.

We can see the following detections in the evidence:

```
Detected Malicious JA3:  807fca46d9d0cf63adf4e5e80e414bbe from source address
2001:718:2:903:b877:48ae:9531:fbfc AS: CESNET z.s.p.o. description:  Tofsee ['malicious']
```

JA3 fingerprint the client part of the SSL certificate. This indicates that the source IP 2001:718:2:903:b877:48ae:9531:fbfc was infected with one of the Tofsee malware family

Slips also detected the connection to the database:

```
SSL certificate validation failed with (certificate has expired) Destination IP:␣
↪2a02:4780:dead:d8f::1. SNI: experimentsas.000webhostapp.com
```

From the RAT analysis, we know that `000webhostapp.com` is the web hosting service used by the C&C server.

Slips also detected

```
Connection to unknown destination port 6669/TCP destination IP 2001:67c:2564:a191::fff:1.
↪ (['open.ircnet.net'])
Connection to unknown destination port 8000/TCP destination IP 192.168.131.1.
```

From the APK list of IRC servers shown in the RAT analysis, we know that the phone connects on port 6669/TCP and 8000/TCP to different IRC servers to receive the malicious commands. The rDNS of the server is also printed in the alert `open.ircnet.net`

Our machine learning module rnn-cc-detection detected the C&C server using recurrent neural network

```
Detected C&C channel, destination IP: 192.168.131.1 port: 8000/tcp score: 0.9871
```

Slips also detected

```
Possible DGA or domain scanning. 192.168.131.2 failed to resolve 15 domains
```

The above detections are evidence that when accumulated, resulted in an alert.

To view all evidence that slips detected including those that weren't enough to generate an alert, you can

```
cat output/alerts.log
```

Slips also has another log file in JSON format so they can be easily parsed and exported. See the exporting section of the documentation.

The generated alerts in this file follow CESNET's IDEA0 format.

```
cat output/alerts.json
```

11.2 Emotet

We will be analysing several Emotet PCAPs starting from infection, until Trickbot and Qakbot malwares are dropped.

The captures contain different actions done by the Emotet and trickbot controller. For detailed analysis, check Paloalto's blog Examining Emotet Infection Traffic.

11.2.1 Emotet infection

We will be analysing this Emotet PCAP download here. password: `infected`

When running Slips on the PCAP

```
./slips.py -f Example-1-2021-01-06-Emotet-infection.pcap
```

We get the following alerts

The reconnection attemps shown in the analysis

are detected by Slips in the following evidence

```
        Detected a connection without DNS resolution to IP: 46.101.230.194
        Detected Multiple reconnection attempts to Destination IP: 46.101.230.194 from␣
↪IP: 10.1.6.206
```

11.2.2 Trickbot

Analyzing the next PCAP download here that contains the Trickbot traffic. password: `infected`

Running slips on the pcap

```
 ./slips.py -f Example-4-2021-01-05-Emotet-infection-with-Trickbot.pcap
```

Slips detects a self signed SSL certificate to 102.164.208.44 which is the trickbot IP associated with data exfiltration

```
        Detected SSL certificate validation failed with (self signed certificate)␣
↪Destination IP: 102.164.208.44
```

Slips also detected

```
Detected a connection without DNS resolution to IP: 102.164.208.44.
```

and

```
Detected Connection to unknown destination port 449/TCP destination IP 102.164.208.44.
```

11.2.3 Qakbot

Analyzing the next PCAP download here that contains the Qakbot traffic. password: `infected`

Running slips on the pcap

```
./slips.py -f Example-5-2020-08-18-Emotet-infection-with-Qakbot.pcap
```

Slips detected that the victim 192.168.100.101 is infected with Qakbo using JA3

```
Detected Malicious JA3: 7dd50e112cd23734a310b90f6f44a7cd from source address 192.168.100.
↪101 description: Quakbot ['malicious']
Detected Malicious JA3: 57f3642b4e37e28f5cbe3020c9331b4c from source address 192.168.100.
↪101 description: Gozi ['malicious']
```

We can also see a Domain generation algorithm detection by the same victim 192.168.100.101

```
Detected possible DGA or domain scanning. 192.168.100.101 failed to resolve 40 domains
```

And an expired certificate to samaritantec.com. This domain was reported as hosting an Emotet binary on the same date.

```
Detected SSL certificate validation failed with (certificate has expired) Destination␣
↪IP: 43.255.154.32. SNI: samaritantec.com
```

Slips then detected

```
        Detected C&C channel, destination IP: 71.80.66.107 port: 443/tcp score: 0.9601
        etected a connection without DNS resolution to IP: 71.80.66.107. AS: CHARTER-
↪20115, rDNS: 071-080-066-107.res.spectrum.com
```

a quick search in virustotal shows that this IP 71.80.66.107 is associated with qakbot

and a port scan

```
Detected horizontal port scan to port 443/TCP. From 192.168.100.101 to 6 unique dst IPs.␣
↪Tot pkts: 21. Threat Level: medium
```

11.3 DroidJack v4.4 RAT

Running Slips on DroidJack v4.4 RAT download here. password: `infected`.

The capture contains different actions done by the RAT controller(e.g. upload a file, get GPS location, monitor files, etc.). For detailed analysis, check Kamila Babayeva's blog Analysis of DroidJack v4.4 RAT network traffic.

From the analysis we know that:

- The controller IP address: 147.32.83.253
- The victim's IP address: 10.8.0.57

When running slips on the PCAP

```
./slips.py -f RAT02.pcap
```

We get the following alerts

Slips detected the connection to the C&C server using an unknown port

```
Detected Connection to unknown destination port 1334/TCP destination IP 147.32.83.253.
```

Slips also detected the reconnection attemps made from the victim to the C&C server

```
Multiple reconnection attempts to Destination IP: 147.32.83.253 from IP: 10.8.0.57
```

Slips also detects connections without resolutions due to their wide usages among malware to either check internet connectivity or get commands fro the C&C servers.

```
        Detected a connection without DNS resolution to IP: 147.32.83.253. AS: CESNET z.
↪s.p.o., rDNS: dhcp-83-253.felk.cvut.cz
```

The indentification (AS, SNI, rDNS) of each IP, if available, is printed in every evidence generated by Slips.

Our Threat intelligence feed Abuse.ch detected a malicious JA3 indicating that the victim 10.8.0.57 is infected

```
Detected Malicious JA3: 7a29c223fb122ec64d10f0a159e07996 from source address 10.8.0.57␣
↪description: ['malicious']
```

And our machine learning models detected the C&C server

```
Detected C&C channel, destination IP: 147.32.83.253 port: 1334/tcp score: 0.9755
```

Slips creates a profile per each IP that appeared in the traffic. Each profile contains flows sent from this IP. Each flow is described with a specific letter which description can be found here.

Considering that, Slips detects the C&C channel over 1334/TCP. Slips' machine learning module called LSTM detecting C&C channel is shown below

Slips did not detect periodic connection over 1337/UDP because the LSTM module focuses on TCP. But from the behavioral model of the connections over 1337/UDP shown below, we can conclude that the model is periodic and most of connections are of a small size.

CHAPTER

TWELVE

CONTRIBUTING

All contributions are welcomed, thank you for taking the time to contribute to this project! These are a set of guidelines for contributing to the development of Slips [1].

12.1 How can you contribute?

- Run Slips and report bugs and needed features, and suggest ideas
- Pull requests with a solved GitHub issue and new feature
- Pull request with a new detection module.

12.2 Persistent Git Branches

The following git branches in the Slips repository are permanent:

- `master`: contains the stable version of Slips, with new versions at least once a month.
- `develop`: contains the latest unstable version of Slips and also its latest features. All new features should be based on this branch.

12.3 Naming Git branches for Pull Requests

To keep the Git history clean and facilitate the revision of contributions we ask all branches to follow concise namings. These are the branch-naming patterns to follow when contributing to Slips:

- author-bugfix-: pull request branch, contains one bugfix,
- author-docs-: pull request branch, contains documentation work,
- author-enhance-: pull request branch, contains one enhancement (not a new feature, but improvement nonetheless)
- author-feature-: pull request branch, contains a new feature,
- author-refactor-: pull request branch, contains code refactoring,

12.4 What branch should you base your contribution to Slips?

As a general rule, base your contributions to the `develop` branch.

12.5 Creating a pull request

Commits:

- Commits should follow the KISS principle: do one thing, and do it well (keep it simple, stupid).
- Commit messages should be easily readable, imperative style ("Fix memory leak in…", not "FixES mem…")

Pull Requests:

- If you have developed multiple features and/or bugfixes, create separate branches for each one of them, and request merges for each branch;
- Each PR to develop will trigger the develop Github checks, these checks will run Slips unit tests and integration tests locally in a ubuntu VM and in docker to make sure the branch is ready to merge.
- PRs won't be merged unless the checks pass.
- The cleaner you code/change/changeset is, the faster it will be merged.

12.6 Beginner tips on how to create a PR in Slips

Here's a very simple beginner-level steps on how to create your PR in Slips

1. Fork the Slips repo
2. Clone the forked repo
3. In your clone, checkout origin/develop: `git checkout origin develop`
4. Install slips pre-commit hooks `pre-commit install`
5. Generate a baseline for detecting secrets before they're committed `detect-secrets scan --exclude-files ".*dataset/.*|(?x)(^config/local_ti_files/own_malicious_JA3.csv$|.*test.*|.*\.md$)" > .secrets.baseline`
6. Create your own branch off develop using your name and the feature name: `git checkout -b <yourname>_<feature_name> develop`
7. Change the code, add the feature or fix the bug, etc. then commit with a descriptive msg `git commit -m "descriptive msg here"`
8. Test your code: this is a very important step. you shouldn't open a PR with code that is not working: `./tests/run_all_tests.sh`
9. If some tests didn't pass, it means you need to fix something in your branch.
10. Push to your own repo: `git push -u origin <yourname>_<feature_name>`
11. Open a PR in Slips, remember to set the base branch as develop.
12. List your changes in the PR description

12.7 Rejected PRs

We will not review PRs that have the following:

- Code that's not tested. a screenshot of the passed tests is required for each PR.
- PRs without steps to reproduce your proposed changes.
- Asking for a step by step guide on how to solve the problem. It is ok to ask us clarifications after putting some effort into reading the code and the docs. but asking how exactly should i do X shows that you didn't look at the code

Some IDEs like PyCharm and vscode have the option to open a PR from within the IDE.

That's it, now you have a ready-to-merge PR!

---

[1] These contributions guidelines are inspired by the project Snoopy

CHAPTER
THIRTEEN

HOW TO CREATE A NEW SLIPS MODULE

13.1 What is SLIPS and why are modules useful

Slips is a machine learning-based intrusion prevention system for Linux and MacOS, developed at the Stratosphere Laboratories from the Czech Technical University in Prague. Slips reads network traffic flows from several sources, applies multiple detections (including machine learning detections) and detects infected computers and attackers in the network. It is easy to extend the functionality of Slips by writing a new module. This blog shows how to create a new module for Slips from scratch.

13.2 Goal of this Blog

This blog creates an example module to detect when any private IP address communicates with another private IP address. What we want is to know if, for example, the IP 192.168.4.2, is communicating with the IP 192.168.4.87. This simple idea, but still useful, is going to be the purpose of our module. Also, it will generate an alert for Slips to consider this situation. Our module will be called `local_connection_detector`.

13.2.1 High-level View of how a Module Works

The Module consists of the init() function for initializations, subscribing to channels, reading API files etc.

The main function of each module is the `main()`, this function is run in a while loop that keeps looping as long as Slips is running so that the module doesn't terminate.

In case of errors in the module, the `main()` function should return 1 which will cause the module to immediately terminate.

any initializations that should be run only once should be placed in the init() function OR the `pre_main()`. the `pre_main()` is a function that acts as a hook for the main function. it runs only once and then the main starts running in a loop. the pre-main is the place for initialization logic that cannot be done in the init, for example dropping the root privileges from a module. we'll discuss this in more detail later.

Each module has a common `print()` method that handles text printing and logging by passing everything to the `OutputProcess.py` for processing. the print function is implemented in the abstract module `slips_files/common/abstracts.py` and used by all modules.

Each Module has its own `shutdown_gracefully()` function that handles cleaning up after the module is done processing. It handles for example:

- Saving a model before Slips stops
- Saving alerts in a .txt file if the module's job is to export alerts
- Telling the main module (slips.py) that the module is done processing so slips.py can kill it etc.

13.3 Developing a Module

When Slips runs, it automatically loads all the modules inside the `modules/` directory. Therefore, our new module should be placed there.

Slips has a template module directory that we are going to copy and then modify for our purposes.

```
cp -a modules/template modules/local_connection_detector
```

13.3.1 Changing the Name of the Module

Each module in Slips should have a name, author and description.

We should change the name inside the py file by finding the lines with the name and description in the class 'Module' and changing them:

```
name = 'local_connection_detector'
description = (
    'detects connections to other devices in your local network'
    )
authors = ['Your name']
```

At the end you should have a structure like this:

```
modules/
├─ local_connection_detector/
│   ├─ __init__.py
│   ├─ local_connection_detector.py
```

The **init**.py is to make sure the module is treated as a python package, don't delete it.

Remember to delete the **pycache** dir if it's copied to the new module using:

`rm -r modules/local_connection_detector/__pycache__`

13.3.2 Redis Pub/Sub

First, we need to subscribe to the channel `new_flow`

```
self.c1 = self.db.subscribe('new_flow')
```

and add this to the module's list of channels

```
self.channels = {
    'new_flow': self.c1,
}
```

this list is used to get msgs from the channel later.

So now everytime slips sees a new flow, you can access it from your module using the following line

```
msg = self.get_msg('new_flow')
```

the implementation of the get_msg is placed in the abstract module in `slips_files/common/abstracts.py` and is inherited by all modules.

The above line checks if a message was received from the channel you subscribed to.

Now, you can access the content of the flow using

```
flow = msg['data']
```

Thus far, we have the following code that gets a msg everytime slips reads a new flow

```
def init(self):
    self.c1 = self.db.subscribe('new_ip')
    self.channels = {
        'new_ip': self.c1,
    }
```

```
  def pre_main(self):
        """
        Initializations that run only once before the main() function runs in a loop
        """
        utils.drop_root_privs()
```

```
 def main(self):
    """Main loop function"""
    if  msg:= self.get_msg('new_flow'):
        #TODO
        pass
```

13.3.3 Detecting connections to local devices

Now that we have the flow, we need to:

- Extract the source IP
- Extract the destination IP
- Check if both of them are private
- Generate an evidence

Extracting IPs is done by the following:

```
msg = msg['data']
msg = json.loads(msg)
flow = json.loads(msg['flow'])
uid = next(iter(flow))
flow = json.loads(flow[uid])
saddr = flow['saddr']
daddr = flow['daddr']
timestamp = flow['ts']
```

Now we need to check if both of them are private.

```
import ipaddress
srcip_obj = ipaddress.ip_address(saddr)
dstip_obj = ipaddress.ip_address(daddr)
if srcip_obj.is_private and dstip_obj.is_private:
    #TODO
    pass
```

Now that we're sure both IPs are private, we need to generate an alert.

Slips requires certain info about the evidence to be able to deal with them.

first, since we are creating a new evidence, other than the ones defined in the EvidenceType Enum in `StratosphereLinuxIPS/slips_files/core/evidence_structure/evidence.py` then, we need to add it

so the EvidenceType Enum in `slips_files/core/evidence_structure/evidence.py` would look something like this

```
class EvidenceType(Enum):
    """
    These are the types of evidence slips can detect
    """
    ...
    CONNECTION_TO_LOCAL_DEVICE = auto()
```

now we have our evidence type supported. it's time to set the evidence!

Now we need to use the Evidence structure of slips, to do that, first import the necessary dataclasses

```
from slips_files.core.evidence_structure.evidence import \
    (
        Evidence,
        ProfileID,
        TimeWindow,
        Victim,
        Attacker,
        ThreatLevel,
        EvidenceType,
        IoCType,
        Direction,
        IDEACategory,
    )
```

now use them,

```
# on a scale of 0 to 1, how confident you are of this evidence
confidence = 0.8
# how dangerous is this evidence? info, low, medium, high, critical?
threat_level = ThreatLevel.HIGH

# the name of your evidence, you can put any descriptive string here
# this is the type we just created
evidence_type = EvidenceType.CONNECTION_TO_LOCAL_DEVICE
# what is this evidence category according to IDEA categories
category = IDEACategory.ANOMALY_CONNECTION
# which ip is the attacker here?
```

(continues on next page)

(continued from previous page)

```
attacker = Attacker(
            direction=Direction.SRC, # who's the attacker the src or the dst?
            attacker_type=IoCType.IP, # is it an IP? is it a domain? etc.
            value=saddr # the actual ip/domain/url of the attacker, in our case, this is␣
↪the IP
        )
victim = Victim(
            direction=Direction.SRC,
            victim_type=IoCType.IP,
            value=daddr,
        )
# describe the evidence
description = f'A connection to a local device {daddr}'
# the current profile is the source ip,
# this comes in the msg received in the channel
# the profile this evidence should be in, should be the profile of the attacker
# because this is evidence that this profile is attacker others right?
profile = ProfileID(ip=saddr)
# Profiles are split into timewindows, each timewindow is 1h,
# this if of the timewindwo comes in the msg received in the channel
twid_number = int(
    msg['twid'].replace("timewindow",'')
    )
timewindow = TimeWindow(number=twid_number)
# how many flows formed this evidence?
# in the case of scans, it can be way more than 1
conn_count = 1
# list of uids of the flows that are part of this evidence
uid_list = [uid]
# no use the above info to create the evidence obj
evidence = Evidence(
                evidence_type=evidence_type,
                attacker=attacker,
                threat_level=threat_level,
                category=category,
                description=description,
                victim=victim,
                profile=profile,
                timewindow=timewindow,
                uid=uid_list,
                # when did this evidence happen? use the
                # flow's ts detected by zeek
                # this comes in the msg received in the channel
                timestamp=timestamp,
                conn_count=conn_count,
                confidence=confidence
            )
self.db.set_evidence(evidence)
```

13.3.4 Testing the Module

The module is now ready to be used. You can copy/paste the complete code that is here

First we start Slips by using the following command:

```
./slips.py -i wlp3s0 -o local_conn_detector
```

-o is to store the output in the `local_conn_detector/` dir.

Then we make a connnection to a local ip (change it to a host you know is up in your network)

```
ping 192.168.1.18
```

And you should see your alerts in ./local_conn_detector/alerts.log by using

```
cat local_conn_detector/alerts.log
```

```
Using develop - 9f5f9412a3c941b3146d92c8cb2f1f12aab3699e - 2022-06-02 16:51:43.989778

2022/06/02-16:51:57: Src IP 192.168.1.18              . Detected a connection to a local␣
↪device 192.168.1.12
2022/06/02-16:51:57: Src IP 192.168.1.12              . Detected a connection to a local␣
↪device 192.168.1.18
```

13.3.5 Conclusion

Due to the high modularity of slips, adding a new slips module is as easy as modifying a few lines in our template module, and slips handles running your module and integrating it for you.

This is the list of the modules Slips currently have. You can enhance them, add detections, suggest new ideas using our Discord or by opening a PR.

For more info about the threat levels, check the docs

Detailed explanation of IDEA categories here

Detailed explanation of Slips profiles and timewindows here

Contributing guidelines

13.4 Complete Code

Here is the whole local_connection_detector.py code for copy/paste.

```
import ipaddress
import json

from slips_files.core.evidence_structure.evidence import \
    (
        Evidence,
        ProfileID,
        TimeWindow,
```

(continues on next page)

(continued from previous page)

```
        Victim,
        Attacker,
        ThreatLevel,
        EvidenceType,
        IoCType,
        Direction,
        IDEACategory,
    )
from slips_files.common.imports import *

class Module(IModule, multiprocessing.Process):
    # Name: short name of the module. Do not use spaces
    name = 'local_connection_detector'
    description = 'detects connections to other devices in your local network'
    authors = ['Template Author']

    def init(self):
        # To which channels do you wnat to subscribe? When a message
        # arrives on the channel the module will wakeup
        # The options change, so the last list is on the
        # slips/core/redis_database.py file. However common options are:
        # - new_ip
        # - tw_modified
        # - evidence_added
        # Remember to subscribe to this channel in redis_db/database.py
        self.c1 = self.db.subscribe('new_flow')
        self.channels = {
            'new_flow': self.c1,
            }

    def shutdown_gracefully(
            self
            ):
        # Confirm that the module is done processing
        self.db.publish('finished_modules', self.name)

    def pre_main(
            self
            ):
        """
        Initializations that run only once before the main() function runs in a loop
        """
        utils.drop_root_privs()

    def main(
            self
            ):
        """Main loop function"""
        if msg := self.get_msg('new_flow'):
            msg = msg['data']
            msg = json.loads(msg)
            flow = json.loads(msg['flow'])
```

(continues on next page)

(continued from previous page)

```
            uid = next(iter(flow))
            flow = json.loads(flow[uid])
            saddr = flow['saddr']
            daddr = flow['daddr']
            timestamp = flow['ts']
            srcip_obj = ipaddress.ip_address(saddr)
            dstip_obj = ipaddress.ip_address(daddr)
            if srcip_obj.is_private and dstip_obj.is_private:
                # on a scale of 0 to 1, how confident you are of this evidence
                confidence = 0.8
                # how dangerous is this evidence? info, low, medium, high, critical?
                threat_level = ThreatLevel.HIGH

                # the name of your evidence, you can put any descriptive string here
                # this is the type we just created
                evidence_type = EvidenceType.CONNECTION_TO_LOCAL_DEVICE
                # what is this evidence category according to IDEA categories
                category = IDEACategory.ANOMALY_CONNECTION
                # which ip is the attacker here?
                attacker = Attacker(
                            direction=Direction.SRC, # who's the attacker the src or the␣
↪dst?
                            attacker_type=IoCType.IP, # is it an IP? is it a domain? etc.
                            value=saddr # the actual ip/domain/url of the attacker, in␣
↪our case, this is the IP
                        )
                victim = Victim(
                            direction=Direction.SRC,
                            victim_type=IoCType.IP,
                            value=daddr,
                        )
                # describe the evidence
                description = f'A connection to a local device {daddr}'
                # the current profile is the source ip,
                # this comes in the msg received in the channel
                # the profile this evidence should be in, should be the profile of the␣
↪attacker
                # because this is evidence that this profile is attacker others right?
                profile = ProfileID(ip=saddr)
                # Profiles are split into timewindows, each timewindow is 1h,
                # this if of the timewindwo comes in the msg received in the channel
                twid_number = int(
                    msg['twid'].replace("timewindow",'')
                    )
                timewindow = TimeWindow(number=twid_number)
                # how many flows formed this evidence?
                # in the case of scans, it can be way more than 1
                conn_count = 1
                # list of uids of the flows that are part of this evidence
                uid_list = [uid]
                # no use the above info to create the evidence obj
                evidence = Evidence(
```

(continues on next page)

(continued from previous page)

```
                                evidence_type=evidence_type,
                                attacker=attacker,
                                threat_level=threat_level,
                                category=category,
                                description=description,
                                victim=victim,
                                profile=profile,
                                timewindow=timewindow,
                                uid=uid_list,
                                # when did this evidence happen? use the
                                # flow's ts detected by zeek
                                # this comes in the msg received in the channel
                                timestamp=timestamp,
                                conn_count=conn_count,
                                confidence=confidence
                            )
                self.db.set_evidence(evidence)
```

13.5 Line by Line Explanation of the Module

This section is for more detailed explanation of what each line of the module does.

```
from slips_files.common.imports import *
```

13.6 This line imports all the common modules that need to be imported by all Slips modules in order for them to work you can check the import here slips_files/common/imports.py

In order to print in your module, you simply use the following line

```
self.print("some text", 1, 0)
```

and the text will be sent to the output queue to process, log, and print to the terminal.

Now here's the pre_main() function, all initializations like dropping root privs, checking for API keys, etc should be done here

```
utils.drop_root_privs()
```

the above line is responsible for dropping root privileges, so if slips starts with sudo and the module doesn't need the root permissions, we drop them.

Now here's the main() function, this is the main function of each module, it's the one that gets executed when the module starts.

All the code in this function is run in a loop as long as the module is online.

in case of an error, the module's main should return non-zero and the module will finish execution and terminate. if there's no errors, the module will keep looping until it runs out of msgs in the redis channels and will call shutdown_gracefully() and terminate.

```
if msg := self.get_msg('new_flow'):
```

The above line listens on the channel called `new_flow` that we subscribed to earlier.

The messages received in the channel are flows the slips read by the the input process.

13.7 Reading Input flows from an external module

Slips relies on input process for reading flows, either from an interface, a pcap, or zeek files, etc.

If you want to add your own module that reads flows from somehwere else, for example from a simulation framework like the CYST module, you can easily do that using the `--input-module <module_name>` parameter

Reading flows should be handeled by that module, then sent to the inputprocess for processing using the `new_module_flow` channel.

For now, this feature only supports reading flows in zeek json format, but feel free to extend it.

13.7.1 How to shutdown_gracefully()

The `stop_message` is sent from the main slips.py to the `control_module` channel to tell all modules that slips is stopping and the modules should finish all the processing it's doing and shutdown.

So, for example if you're training a ML model in your module, and you want to save it before the module stops,

You should place your save_model() function in the shutdown_gracefully() function, right before the module announces its name as finished in the `finished_modules` channel

Inside shutdown_gracefully() we have the following line, This is the module, responding to the stop_message, telling slips.py that it successfully finished processing and is ready to be killed.

```
self.db.publish('finished_modules', self.name)
```

13.7.2 Troubleshooting

Most errors occur when running the module inside SLIPS. These errors are hard to resolve, because warnings and debug messages may be hidden under extensive outputs from other modules.

If the module does not start at all, make sure it is not disabled in the config/slips.conf file. If that is not the case, check that the __init__.py file is present in module directory, and read the output files (errors.log and slips.log) - if there were any errors (eg. import errors), they would prevent the module from starting.

In case that the module is started, but does not receive any messages from the channel, make sure that:

```
-The channel is properly subscribed to in the module

-Messages are being sent in this channel

-Other modules subscribed to the channel get the message
```

(continues on next page)

(continued from previous page)

```
- the channel name is present in the supported_channels list in redis_db/database.py
```

13.7.3 Testing

Slips has 2 kinds of tests, unit tests and integration tests.

integration tests are done by testing all files in our `dataset/` dir and are done by the test files in `tests/integration_tests/`

Before pushing, run the unit tests and integration tests by:

1- Make sure you're in slips main dir (the one with kalipso.sh)

2- Run all tests `./tests/run_all_tests.sh`

Slips supports the -P flag to run redis on your port of choice. this flag is used so that slips can keep track of the ports it opened while testing and close them later.

13.7.4 Adding your own unit tests

Slips uses `pytest` as the main testing framework, You can add your own unit tests by:

1- create a file called `test_module_name.py` in the `tests/` dir

2- create a method for initializing your module in `tests/module_factory.py`

3- every function should start with `test_`

4- go to the main slips dir and run `./tests/run_all_tests.sh` and every test file in the `tests/` dir will run

13.7.5 Getting in touch

Feel free to join our Discord server and ask questions, suggest new features or give us feedback.

PRs and Issues are welcomed in our repo.

13.7.6 Conclusion

Adding a new feature to SLIPS is an easy task. The template is ready for everyone to use and there is not much to learn about Slips to be able to write a module.

If you wish to add a new module to the Slips repository, issue a pull request and wait for a review.

CHAPTER

FOURTEEN

FAQ

14.1 Slips starting too slow in docker

Make sure you're not running many containers at the same time because they share kernel resources even though they're isolated.

14.2 Getting "Illegal instruction" error when running slips

If the tensorflow version you're using isn't compatible with your architecture, you will get the "Illegal instruction" error and slips will terminate.

To fix this you can disable the modules that use tensorflow by adding `rnn-cc-detection, flowmldetection` to the `disable` key in `config/slips.conf`

14.3 Docker time is not in sync with that of the host

You can add your local /etc/localtime as volume in Slips Docker container by using:

```
docker run -it --rm --net=host --cap-add=NET_ADMIN -v /etc/localtime:/etc/localtime:ro --
↪name slips stratosphereips/slips:latest
```

CHAPTER

FIFTEEN

CODE DOCUMENTATION

15.1 How Slips Works

1. slips.py is the entry point, it's responsible for starting all modules, and keeping slips up until the analysis is finished.
2. slips.py starts the input process, which is the one responsible for reading the flows from the files given to slips using -f it detects the type of file, reads it and passes the flows to the profiler process. if slips was given a PCAP or is running on an interface , the input process starts a zeek thread that analyzes the pcap/interface using slips' own zeek configuration and sends the generated zeek flows to the profiler process.
3. slips.py also starts the update manager, it updates slips local TI files, like the ones stored in slips_files/organizations_info and slips_files/ports_info. later, when slips is starting all the modules, slips also starts the update manager but to update remote TI files in the background in this case.
4. Once the profiler process receives the flows read by the input process, it starts to convert them to a structure that slips can deal with. it creates profiles and time windows for each IP it encounters.
5. Profiler process gives each flow to the appropriate module to deal with it. for example flows from http.log will be sent to http_analyzer.py to analyze them.
6. Profiler process stores the flows, profiles, etc. in slips databases for later processing. the info stored in the dbs will be used by all modules later. Slips has 2 databases, Redis and SQLite. it uses the sqlite db to store all the flows read and labeled. and uses redis for all other operations. the sqlite db is created in the output directory, meanwhite the redis database is in-memory. 7-8. using the flows stored in the db in step 6 and with the help of the timeline module, slips puts the given flows in a human-readable form which is then used by the web UI and kalipso UI.
7. when a module finds a detection, it sends the detection to the evidence process to deal with it (step 10) but first, this evidence is checked by the whitelist to see if it's whitelisted in our config/whitelist.conf or not. if the evidence is whitelisted, it will be discarded and won't go through the next steps
8. now that we're sure that the evidence isn't whitelisted, the evidence process logs it to slips log files and gives the evidence to all modules responsible for exporting evidence. so, if CEST, Exporting modules, or CYST is enabled, the evidence process notifies them through redis channels that it found an evidence and it's time to share the evidence.
9. if the blocking module is enabled using -p, the evidence process shares all detected alerts to the blocking module. and the blocking module handles the blocking of the attacker IP through the linux firewall (supported in linux only)
10. if p2p is enabled in config/slips.conf, the p2p module shares the IP of the attacker, its' score and blocking requests sent by the evidence process with other peers in the network so they can block the attackers before they reach them.

11. The output process is slips custom logging framework. all alerts, warnings and info printed are sent here first for proper formatting and printing.

This is a brief explanation of how slips works for new contributors.

All modules described above are talked about in more detail in the rest of the documentation.

15.2 Code Docs

Slips auto-generated code documentation here