Ben Elam
bae53@msstate.edu
CSE 6243
October 31, 2016

Overview

To fulfill the project requirements, I created peekapp, a packet-monitoring IDS layer, on top of the Scapy python package.

Features

As specified in the project requirements, peekapp will detect, log, and alert on:

- Any traffic to or from blacklisted IP addresses
- DNS requests for blacklisted domain names
- Unencrypted web traffic to URLs containing a set of blacklisted strings
- TCP or UDP payloads containing any of a set of simple signatures
- Network port scanning activity

Installation and Basic Usage

Peekapp requires Python 2.7, and the use of a virtual environment is highly recommended. To install, unzip the archive and do the following:

```
$ cd peekapp
$ python setup.py install
$ peekapp --help
Usage: peekapp [OPTIONS] COMMAND [ARGS]...

  peekapp is a packet-monitoring IDS layer.

Options:
  -d, --domain-blacklist FILENAME
                                  File listing domains for which any DNS
                                  traffic should trigger an alert
  -u, --url-blacklist FILENAME    File listing strings triggering an alert
                                  when present in URLs
  -i, --ip-blacklist FILENAME     File listing IPs for which any traffic
                                  should trigger an alert
  -s, --signatures FILENAME       File listing TCP/UDP payload signatures
  -l, --logfile FILENAME          Output file to which logged packets are
                                  appended
  -a, --alert-timeout FLOAT       Minutes to wait before summarizing redundant
                                  alerts
  -p, --port-scan                 Flag to enable port scan detection
  --help                          Show this message and exit.

Commands:
  iface  Monitor a specific network interface
  pcap   Perform static analysis on a PCAP dump
```

As shown above, `peekapp --help` will print an overview of arguments and subcommands. A log file must be specified, and no packets will be logged without one or more blacklist files specified as options. Finally, a subcommand, either `iface` or `pcap` must be provided in order to monitor a network interface or a PCAP-format packet dump, respectively. The interface or packet dump to be analyzed is to be specified as the only argument to the selected subcommand.

To use the `iface` subcommand, peekapp must be run with root privileges. This makes the interface slightly cumbersome if peekapp is installed to a virtual environment or Python installation that is not normally in root's Python path. However, it is certainly preferable to installing peekapp via the system Python installation.

Examples:

```
# Execute peekapp with root privileges using installed executable
$ sudo `which peekapp` -l out.log -d bad_domains.cfg -s sketchy_signatures.cfg
iface wlan0

# Execute included script using Python install of the current (non-root) user
$sudo `which python` peekapp.py -l out.log -d bad_domains.cfg -s
sketchy_signatures.cfg iface wlan0

# Static analysis is a bit less complicated
$peekapp -l out.log -i ips_that_should_be_reserved.cfg pcap oldtraffic.pcap

# Port scan detection on network interface
$sudo `which peekapp` -p -l out.log iface eth0
```

Configuration

With the exception of port scan detection, each of peekapp's features is enabled at the command line by passing the name of a file with the feature's option flag. Each line of the file should specify a single rule by which peekapp can identify packets of interest. For instance, each line of the file whose name is passed to `-s` (or `--signatures`) should be a string such that any packet containing the string in its signature is a target for capture.

Note that arbitrary byte sequences may be used when specifying a configuration file listing payload signatures. Non-character bytes must be specified via their two-digit hex encoding, the format of which must comply with Python specification PEP 223 [4]. In short, record the two-digit hex number `HH` as `\xHH`, and characters which would otherwise require escaping in a Python string must be prefaced with a backslash, e.g. `abc\'def`.

There are currently three other command line options available. The name of a log file must be specified to the `-l` flag. Port scan detection only requires for the `-p` flag to be present; the flag accepts no arguments. Finally, the `-a` flag can be used to specify the length of the delay, in minutes, which peekapp should place between the first time a particular kind of packet is produces an alert and the time at which any subsequent packets of the same kind are summarized to produce a second alert.

Report

Decision Process & Design

My entire networking background consists of what I've learned in the course of studying for this class. Hence, I chose the easy route and went with Python and Scapy [0, 1].

I also used a couple of third-party libraries to ease the development of the CLI and the test suite. I opted for the Click [5] module, instead of the argparse module in Python's standard library, due to personal familiarity with Click. It's treated me well in the past, and I still recommend it, but this choice really hurt me, as I describe below. Similarly, I went with PyTest [3] instead of the stock Python choice, UnitTest, out of familiarity and because it allows for short, clear tests.

The official Scapy documentation is somewhat scarce, so I started reading help files from within an interactive session. When I found that many classes and methods were undocumented, I decided to just dive into the project's source. Although most of the code I found was devoid of docstrings and other comments, the code itself was very readable and served as my primary reference.

When I came across Scapy's `pipetool` and `scapypipes` modules [2], I was confused. The classes in `pipetool`, along with its one test function, suggested exactly the system I had sketched out when I received the assignment. The `_ConnectorLogic` class provides a very simple DSL for defining a push-based traffic-analysis program as a directed acyclic graph. However, neither file had been updated in years, no other file in the project imported them, and no issues mentioned them.

Difficulties

I decided on using both `scapy.pipetool` and Click prior to having much in the way of working code or tests. I spent a number of hours drafting the skeleton of the application and reading Scapy's source code before I had a minimally testable product. When I finally reached that point late Wednesday night, I found that something was very wrong.

For some reason, Click sent the `interface` argument to `sniff` multiple times when calling `peekapp.interface.iface`. The first time, `interface` was passed - correctly - as a string. The second and third times, however, it was passed as unicode.

To procure an L2 socket to listen on, Scapy would eventually pass this argument into `struct.pack`, which would crash everything since it expected `<type 'string'>`, not `<type 'unicode'>`. This was rather difficult to discover, however, due to the threading provided by `scapy.pipetool.PipeEngine`, which resulted in unhelpful error messages.

To find the bug, I ultimately wrote my own pipes module on top of `scapy.pipetool._ConnectorLogic`. I was pretty happy with the results until I moved the new code from my test script back into my application code and was faced with the same error message. I was finally able to debug what was happening in `scapy.arch.common.get_if`, link it back to Click, and move on.

At this point, it was Saturday afternoon, and I'd been at an impasse for three days. I decided to keep my own pipes module both for the sake of time and to avoid debugging around `PipeEngine`'s threads. I would enjoy forking the project at a later date to again try using

`scapy.pipetool`, but doing so would require a rewrite of several components, such as `peekapp.alerts`.

Despite getting an early start (see git logs,) this greatly set me back in tackling the actual features of the project. On the one hand, I could have tried implementing each feature independently and glued it all together somehow. Maybe there's a lesson to be learned here about priorities or something. I would have preferred to have the extra time to build a Vagrant testing environment, add TravisCI support for testing, and start on port scan detection prior to 11:30 on Sunday night. On the other hand, I was very happy with the architecture I had sketched out, and I'm even happier now that it's functional. As I've been hacking away on the actual features, I've found the underlying application very nicely structured for accommodating new changes.

Limitations

I assume peekapp would perform better with threading support, which I lost when I abandoned `PipeEngine`, since logged packets will be doubly IO-bound.

I'm pleased with the current alert system for monitoring in-flight packets. However, this system's aggregation model is very much dependent on the timestamps of packets, as well as the order in which they are received. Hence, it is ill-suited for static analysis of a PCAP file via the pcap command. (In particular, it is effectively a differently styled log during static analysis.) A stateless aggregation strategy of all packets within the file would be more appropriate. A parameterizable split-apply-combine strategy for summarizing static logs would make for a good feature addition.

After the aforementioned complications, I started port scan detection very late on Sunday evening, so my implementation is remarkably naive... and, as of yet, untested. It watches for SYN stealth scans between any two devices, not just the host computer. Given a target packet count (tolerance) and a timeout length **t**, peekapp maintains a list of packets with timestamps occurring no more than **t** seconds prior to the timestamp of the latest packet. If a particular cache of packets exceeds the specified tolerance, the cache is bundled into a single record and merged with the other logs. Given this design, peekapp makes the horrifying assumption that packets are arriving in order, as dropping this assumption would make for a more complex computation or a more involved design.

Testing

To test the basic functionality of the application, I first implemented logging of DNS queries. Then, I built the alert system by trial and error, checking for UDP packets bound for a small number of websites. For instance, I don't read CNN, so I assumed my browser wouldn't have any CNN assets cached. Their website also hosts a number of subdomains, which was of interest for testing. Hence, I listed `cnn.com` in a file called `domains.cfg`, then ran `sudo `which python` peekapp.py -l out.log -d domains.cfg iface wlan0`. Once peekapp was running, I opened CNN.com and clicked on anywhere between 6 and 20 articles.

Once I was ready to add additional features, I implemented a small number of functional tests for regression detection using the PyTest module. Each feature (excepted port scan detection) is tested against a PCAP file I produced using Wireshark. Having prior knowledge of the features of each file produced (documented in `peekapp/tests/files/README.md`,) I was able to write tests specific to the data available. Features are required to log the correct number of "bad"

packets, and all logged packets are checked for compliance with the provided traffic rule(s). After installation, the test suite can be run from the topmost source directory via `python setup.py test`.

As of yet, I have done no testing under significant loads, nor have I tested peekapp's port scan detection. Instead of building a virtual machine for this purpose, my original plan was to simply deploy to a Digital Ocean server and provide external traffic via Scapy from my workstation.

Works Cited

[0] Biondi, Philippe. “Scapy Documentation.” Accessed October 30, 2016, secdev.org/projects/scapy/files/scapydoc.pdf.

[1] Biondi, Philippe and the Scapy community. “Scapy v2.1.1-dev documentation.” Accessed October 30, 2016, secdev.org/projects/scapy/doc.

[2] Biondi, Philippe and the Scapy community. “SecDev/Scapy.” Accessed October 30, 2016, github.com/secdev/scapy.

[3] Krekel, Holger and pytest-dev team. “PyTest Documentation.” Accessed October 30, 2016, docs.pytest.org.

[4] Peters, Tim. “PEP 223 -- Change the Meaning of \x Escapes.” Accessed October 30, 2016, python.org/dev/peps/pep-0223.

[5] Ronacher, Armin. “Click Documentation (5.0).” Accessed October 30, 2016, click.pocoo.org.