Welcome!

A Beginner's Guide to Adding New Features to Suricata
Quick Housekeeping

- Webinar is being recorded
- It will be posted on our YouTube channel in the next day or so - https://www.youtube.com/c/OISFSuricata
- Excited for questions - use the chat
- Thank you for being here!
Who are we?

A 501(c) US nonprofit organization missioned to provide architecture and infrastructure to open source security software communities and to projects like Suricata.

Funding comes from financial contributions from world-class security organizations using Suricata and committed open source technologies.
A bit more...

- Our consortium -> we offer several ways to support Suricata's development roadmap and growing community: details at https://oisf.net/ or contact us at info@oisf.net

- Connect with the community -> https://forum.suricata.io/

- SuriCon 2021 -> October 20-22, 2021 in Boston, MA - https://suricon.net/ - call for talks and training proposals is now OPEN
Adding initial support for the PostgreSQL protocol in Suricata

An ongoing project

Juliana Fajardini Reichow
Outreachy intern

suricata-ids.org
Whoami

Brazilian, B. degree in Information Systems, returning to the IT field

- Robot Soccer & AI (C/ C++)
- Customer Development
- Translation/ Proofreading
- Customer & technical support
- Application Layer Protocols (Rust)


Outreachy internships are,

- **Paid** - $6,000 USD total internship stipend
- **Remote** - both interns and mentors work remotely
- **3 months** - internships run twice a year, May - Aug / Dec - Mar
- **Goal** - increase diversity in open source

More info - outreachy.org // @outreachy (twitter)
Contribution phase: how does it work?

- Project choice - affinity, skills, learning goals
- Respect and follow community guidelines
- Read Suricata contribution guidelines
- Accept Contribution Agreement
- Do your best to understand things, pay attention to details... but don’t get stuck:
  - Mentors and community are there to help
- Share contributions via Pull Requests on GitHub
  - Add feedbacks in a different branch
- Help others where you can :)
Redmine Feature #4241

“Add support for the PostgreSQL protocol. Not only will this add support for PostgreSQL but other databases that use the PostgreSQL wire format like CockroachDB, and I believe there are others.”

Initial basics should include support for:
- connection username and database
- queries
Protocol support: PostgreSQL in Rust

Why Rust?
Reliable systems language, strongly focused on being memory safe, without impacting performance (it doesn’t have a garbage collector).

Bonus: Rust nom crate.

Spoiler alert: LOTS of compile errors.
Disclaimer: adding a protocol to Suricata is not a small or beginner task. If you want to start contributing to Suricata, it is advisable to try smaller ones, first.
Protocol... support?

Some key elements:

- Protocols -> Specifications, guidelines
- Network traffic (Communication, exchange)
  - Layers
- Application Layer Protocol - communication between applications
- PostgreSQL - Open Source Database
  - Database Communication
    - Client (Frontend) <-> Server (Backend)
- Parsing
  - analyzing, understanding
- Integrating
Protocol… support?

This means… Providing Suricata with the ability to recognize and understand a given Protocol.

Right now, running a “clean Suricata” (just cloned from GitHub) on a PostgreSQL pcap doesn’t show anything special.

Because Suricata doesn’t “know” PostgreSQL, yet.
Protocol support: suricata.yaml file

Before:

```yaml
## Step 4: App Layer Protocol configuration
#
# Configure the app-layer parsers. The protocol's section details each protocol.
#
# The option "enabled" takes 3 values - "yes", "no", "detection-only". "yes" enables both
detection and the parser, "no" disables both, and "detection-only" enables protocol
detection only (parser disabled).
app-layer:
  protocols:
    newprotoaplayertemplate:
      enabled: yes
    rfb:
      enabled: yes
detection-ports:
      dp: 5900, 5901, 5902, 5903, 5904, 5905, 5906, 5907, 5908, 5909
    # MQTT, disabled by default.
E486: Pattern not found: postgresql
```

After:

```yaml
## Step 4: App Layer Protocol configuration
#
# Configure the app-layer parsers. The protocol's section details each protocol.
#
# The option "enabled" takes 3 values - "yes", "no", "detection-only". "yes" enables both
detection and the parser, "no" disables both, and "detection-only" enables protocol
detection only (parser disabled).
app-layer:
  protocols:
    postgresql:
      enabled: yes
detection-ports:
      dp: 5432
```
Useful resources

Official documentation
- Suricata project page on Redmine:
  - Suricata Installation (advanced guide)
  - Suricata Developers Guide
- Suricata documentation: https://suricata.readthedocs.io/
- Is there an RFC for the protocol? Official manuals

GitHub: Guides, discussions, previous contributions & implementations
- Suricata developer’s guide
- Commits on GitHub
Useful resources: GitHub - doc/devguide

extending/app-layer/parser.rst

Callbacks

The API calls callbacks that are registered at the start of the program.

The function prototype is:

```c
typedef AppLayerResult (*AppLayerParserFPtr)(Flow *f, void *protocol_state,
        AppLayerParserState *pstate,
        const uint8_t *buf, uint32_t buf_len,
        void *local_storage, const uint8_t *flags);
```

Examples

A C example:

```c
static AppLayerResult HTPHandleRequestData(Flow *f, void *htp_state,
        AppLayerParserState *pstate,
        const uint8_t *input, uint32_t input_len,
        void *local_data, const uint8_t *flags);
```

In Rust, the callbacks are similar.

```rust
#[no_mangle]
pub extern "C" fn rs_dns_parse_response_tcp(_: Flow, core::Flow,
    state: std::os::raw::c_void,
    _pstate: std::os::raw::c_void,
    input: std::os::raw::c_void,
    input_len: u32,
    _data: std::os::raw::c_void,
    _flags: u8)
-> AppLayerResult
```

Return Types

Parsers return the type `AppLayerResult`.

There are 3 possible results:
- APP_LAYER_OK - parser consumed the data successfully
- APP_LAYER_ERROR - parser encountered a unrecoverable error
- APP_LAYER_INCOMPLETE(c,n) - parser consumed c bytes, and needs n more before being called again

Rust parsers follow the same logic, but can return
- `AppLayerResult::ok()`
- `AppLayerResult::err()`
- `AppLayerResult::incomplete(c,n)`

For i32 and bool, Rust parsers can also use `.into()`.
Useful resources: GitHub - other commits

**rust: Add DCERP parser**

This parser rewrites the DCE/RPC protocol implementation of Suricata in Rust. More tests have been added to improve the coverage and some fixes have been made to the tests already written in C. Most of the valid tests from C have been imported to Rust.

**File anatomy**

`src/dcerpc.rs`

This file contains the implementation of single transactions in DCE/RPC over TCP. It takes care of REQUEST, RESPONSE, BIND and BINDACK business logic before and after the data parsing. DCERPState holds the state corresponding to a particular transaction and handles all important aspects. It also defines any common structures and constants required for DCE/RPC parsing irrespective of the carrier protocol.

**Implementation**

When a batch of input comes in, there is an analysis of whether the input header and the direction is appropriate. Next check is about the size of fragment. If it is as defined by the header, process goes through else the data is buffered and more data is awaited. After this, type of record as indicated by the header is checked. A call to the appropriate handler is made. After the handling, State is updated with the latest information about whatever record came in. AppLayerResult::ok() is returned in case all went well else AppLayerResult::err() is returned indicating something went wrong.

```
master (#4958) suricata-6.0.1 -> suricata-6.0.0-beta1

inashivb authored and victorjulien committed on May 19, 2020
```

[Official Site](suricata-ids.org)
Useful resources

Tools and resources for packet capture (files: pcaps)
- Wireshark & Wireshark Wiki
- tcpdump (to generate our own)

Other implementations
- Wireshark dissectors…

Suricata Scripts (suricata/scripts/)
- setup-app-layer.py - a great tool when starting with a new protocol!
### Wireshark

#### Display Filter

Filter: `tcp` & `destination eq 10.16.1.10`

#### TCP Packet Details

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>10.16.1.11</td>
<td>10.16.1.10</td>
<td>TCP</td>
<td>74</td>
<td>Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1</td>
</tr>
<tr>
<td>2</td>
<td>0.000299</td>
<td>10.16.1.10</td>
<td>10.16.1.10</td>
<td>TCP</td>
<td>74</td>
<td>Seq=0 Ack=1 Win=29400 Len=0 MSS=1460</td>
</tr>
<tr>
<td>3</td>
<td>0.000624</td>
<td>10.16.1.10</td>
<td>10.16.1.10</td>
<td>TCP</td>
<td>66</td>
<td>Seq=0 Ack=1 Win=29312 Len=0 TSeq=40047445.</td>
</tr>
</tbody>
</table>

#### Packet Details

- Acknowledgment number: 1
- Acknowledgment number (raw): 1750068410
- Flags: 0x018 (PSH, ACK)
- Window size value: 229
- [Calculated window size: 29312]
- [Window size scaling factor: 128]
- Checksum: 0xcb75 [unverified]
- Urgent pointer: 0
- Options: 12 bytes, No-Operation (NOP), No Urgent
- [SEQ/ACK analysis]
- [Timestamps]

#### Protocol Preferences

- **TCP Stream**: Ctrl+Alt+Shift+T
- **UDP Stream**: Ctrl+Alt+Shift+U
- **TLS Stream**: Ctrl+Alt+Shift+S
- **HTTP Stream**: Ctrl+Alt+Shift+H
- **HTTP2 Stream**: Ctrl+Alt+Shift+2
- **QUIC Stream**: Ctrl+Alt+Shift+Q

#### Packets Displayed

- Packets: 10
- Displayed: 10
- Profile: Default
Protocol support: peeking pcaps

suricata/rust/src/applayertemplate/template.pcap (detail)
Possible Approach

- Communication is key!
  - Ask first: “Can I claim this?”
    - “Would this be a good feature?”
  - Share early

- Understand the codebase (basics)
  - codebase
  - folder structure
  - naming conventions
  - resources
  - approaches
Possible Approach

● Understand the protocol
  ○ characteristics & behavior
  ○ coverage
  ○ packets and messages anatomy

● Understand the task
  ○ what is a minimum viable “product”?
  ○ break it into smaller, more approachable tasks.
  ○ what are the building blocks?
    ■ message types (always check PCAPs!)
Getting started

Running suricata on the pcap file without NewProto:

```
$ sudo suricata -c suricata.yaml -r rust/src/applayertemplate/template.pcap -v
```
Getting started
Auto-generating app-layer code for C and/or Rust

From Suricata directory, you can run setup-app-layer.py for generating c or c/rust structure for a new protocol:

```
$ ./scripts/setup-app-layer.py --rust NewProtocol
```
Auto-generating app-layer code for C and/or Rust

Running suricata on the pcap file with NewProto:

```
12:04:41 - (newprotoaplayertemplate.rs:151) Request: Hello World!
12:04:41 - (newprotoaplayertemplate.rs:200) Request: Some("Hello World!")
12:04:41 - (newprotoaplayertemplate.rs:202) Request: Some("Bye")
```

For more information please see:

make[1]: Leaving directory `/home/jufajardini/Workspace/OISF/suricata'
jufajardini@jubassaur:/Workspace/OISF/suricata$ sudo /usr/local/bin/suricata -c /usr/local/etc/suricata/suricata.yml -v -l eth0

The Emerging Threats Open rules are now installed. Rules can be updated and managed with the suricata-update tool.
Challenges

- Coping with large scoped tasks
- Learning curves!
  - Suricata codebase
  - Rust learning curve
  - nom crate learning curve
  - Protocol implementation learning curve

Surf the waves /o/
A few useful steps

- Get an overall understanding of the application layer protocol
  - Look at documentation, find pcaps, inspect them
- Get familiar with Suricata codebase,
- Play with setup-app-layer.py script
- Use it to generate a NewProtocol, run suricata over the template.pcap, start improving and adding from there
A few useful steps

- Find a simple use case, and find or generate pcaps for it
- Write parsers for that
  - Write unit tests
- Do small adjustments that make Suricata recognize the new protocol messages, instead of the template one.
- `rust/src/applayer.rs:: pub struct RustParser is a friend.`
  - There will be one instance of it in the NewProtocol.rs file
An SSL Handshake, on PostgreSQL

Source: Postgres on the wire presentation on PGSon 2014, by Jan Urbánski
SSL Handshake on PostgreSQL (Wireshark peek)
Protocol support: a peek at some pcaps

Suricata parsing an SSL Handshake

SSL Request

SSL Response
What now?

- Working Minimum Viable “Product”
- Parsing Error/Notice response messages
- The missing bits (and bytes) are the next challenge
  - Keep adding to the protocol: more messages and subprotocols
  - TCP gaps handling
  - Keep adding tests and debugging
- Open source project means contribution doesn’t have to end <3
Questions are welcome!

@increible_ju
@jufajardini
jufajardini
jufajardini.wordpress.com

Thank you!
A Beginner’s Guide to Adding New Features to Suricata

Tharushi Jayasekara
Outreachy Intern at OISF
About me

 проведене в Шри-Ланке

 Outreachy Intern at OISF

 Computer Science at University of Colombo

 Absolutely love data structures, algorithms, programming (in Python!)

 .... and most recently InfoSec!
My internship project

Improve Python tools for Suricata
What is Suricata?

- A free and open source, mature, fast and robust network threat detection engine
- Capable of
  - Real time intrusion detection (IDS),
  - Inline intrusion prevention (IPS),
  - Network security monitoring (NSM) and
  - Offline pcap processing
Suricata

Suricata is a free and open source, mature, fast and robust network threat detection engine.

The Suricata engine is capable of real time intrusion detection (IDS), inline intrusion prevention (IPS), network security monitoring (NSM) and offline pcap processing.

Suricata inspects the network traffic using a powerful and extensive rules and signature language, and has

UPCOMING TRAININGS

EVENTS

Webinar - A Beginner's Guide to Adding New Features to Suricata
SuriCon
Suricata-Update & Suricata-Verify

- Most used and active python tools
  - Suricata-Update
    - To update and manage rules for Suricata
    - Used mostly by users
  - Suricata-Verify
    - To test many features of Suricata
    - Used mostly by developers
Suricata-Update & Suricata-Verify

- Most used and active python tools
  - **Suricata-Update**
    - To update and manage rules for Suricata
    - Used mostly by users
  - **Suricata-Verify**
    - To test many features of Suricata
    - Used mostly by developers
What is Suricata Verify?

- A python tool
- Used for QA during development
- To streamline testing many features of Suricata

Suricata Verification Tests

- Tests that run Suricata with a specific configuration and/or inputs and verify the outputs
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>inashlib Check libnss required for JA3 support</td>
<td>tests: fix test requirements for 4.1.x and 4.0.x</td>
<td>2 years ago</td>
</tr>
<tr>
<td>alert-distance-within-1</td>
<td>tests: test that triggers a rule with established though 3whs missing</td>
<td>6 months ago</td>
</tr>
<tr>
<td>alert-no-3whs-established</td>
<td>tests: add tcp async testmyids test</td>
<td>2 months ago</td>
</tr>
<tr>
<td>alert-testmyids-async</td>
<td>tests: add testmyids variant with midstream</td>
<td>17 months ago</td>
</tr>
<tr>
<td>alert-testmyids-midstream</td>
<td>tests: add more midstream test variants</td>
<td>2 months ago</td>
</tr>
<tr>
<td>alert-testmyids-midstream3</td>
<td>tests: add more midstream test variants</td>
<td>2 months ago</td>
</tr>
<tr>
<td>alert-testmyids-midstream5</td>
<td>tests: add more midstream test variants</td>
<td>2 months ago</td>
</tr>
<tr>
<td>alert-testmyids-not-established</td>
<td>setup/check relative to output directory</td>
<td>3 years ago</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
<td>Last Updated</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>inashivb</td>
<td>Check libnss required for JA3 support</td>
<td>12 days ago</td>
</tr>
<tr>
<td>.github/workflows</td>
<td>github-ci: 4.1 is eol, 6.0 test added</td>
<td>2 months ago</td>
</tr>
<tr>
<td>etc</td>
<td>add suricata 4.0.3 default config</td>
<td>3 years ago</td>
</tr>
<tr>
<td>tests</td>
<td>Check libnss required for JA3 support</td>
<td>12 days ago</td>
</tr>
<tr>
<td>util</td>
<td>put all tests in a tests/ directory</td>
<td>3 years ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Move http-evader-* tests to http-evader folder</td>
<td>2 years ago</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>travis: add check for pcap-ng files</td>
<td>2 years ago</td>
</tr>
<tr>
<td>README.md</td>
<td>framework: adds feature filtering per check</td>
<td>18 days ago</td>
</tr>
<tr>
<td>clean.sh</td>
<td>clean: test that we are in a verify directory</td>
<td>3 years ago</td>
</tr>
<tr>
<td>createst.py</td>
<td>createst: add strictctsum option</td>
<td>6 months ago</td>
</tr>
<tr>
<td>run.py</td>
<td>framework: adds feature filtering per check</td>
<td>18 days ago</td>
</tr>
<tr>
<td>travis.sh</td>
<td>travis: add check for pcap-ng files</td>
<td>2 years ago</td>
</tr>
</tbody>
</table>
Usage of SV

- Run tests

From your Suricata source directory run:

```bash
../path/to/suricata-verify/run.py
```
Contribution

1. Implement Parallel Tests Capability
2. Validate output using JSON Schema
3. Support symlink for test directories
4. Convert ‘cmp’ tests to Python
1. Parallel Tests Capability - Before

- Earlier, SV tests were run in **serial**

```
tharushi@tharushi-Inspiron-5567:/suricata$  
real  1m28.030s
user  0m55.463s
sys   0m27.517s
```
1. Parallel Tests Capability - After

- But now they run in **parallel** using multiprocessing
- Number of parallel jobs = Number of cores in your machine

```
Number of concurrent jobs: 4
OK
OK
OK

real  1m28.030s
user  0m55.463s
sys   0m27.517s
```

Before

```
real  0m45.045s
user  1m23.006s
sys   0m35.359s
```

After
1. Parallel Tests Capability - Problems

Problems

1. Too many parallel tests on many cores
   - Limit the maximum number of parallel jobs to 8

2. Failing tests due to timing issues
   - Inherent bug in Suricata!
2. Validate output using JSON Schema

- JSON schema is a specification to define the structure of JSON data

  Eg: `src_ip` is of type `integer`

- Validated the eve.json output, which outputs alerts, anomalies, metadata, file info and protocol specific records using a JSON schema
2. Validate output using JSON Schema

- After each test is run, its output is validated using the schema

```bash
tharushi@tharushi-Inspiron-5567:~/suricata/suricata$ ../suricata-verify/run.py
Number of concurrent jobs: 4
==> alert-distance-within-1: OK
==> alert-no-3whs-established: OK
==> alert-testmyids-async: OK
==> alert-testmyids: OK
==> alert-testmyids-async: FAILED: Invalid JSON schema
==> alert-no-3whs-established: FAILED: Invalid JSON schema
==> alert-testmyids: FAILED: Invalid JSON schema
==> alert-distance-within-1: FAILED: Invalid JSON schema
==> alert-testmyids-midstream5: OK
==> alert-testmyids-midstream3: OK
==> alert-testmyids-midstream: OK
```
2. Validate output using JSON Schema

- Optionally, the validation could be run separately on all of the tests

- Or even just a specific file or directory if you wish to!
3. Support symlink for test directories

- Currently SV does not pick up on symlinks
- Added support for symlinks
Finally

How can you contribute to Suricata?
Open Information Security Foundation

Home

Suricata


Suricata is licensed under the GPLv2. A copy of this license is available in this tarball, or at:
(http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt)

The redmine project page is here: https://redmine.openinfosecfoundation.org/projects/suricata

Suricata-Update

Suricata-update is a rule updater for Suricata.

The redmine project page is here: https://redmine.openinfosecfoundation.org/projects/suricata-update
Things to takeaway

1. Register on Redmine
2. Claim a ticket
3. Assign ticket to yourself
4. Read the contributing guidelines on
   https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Contributing
5. Sign the Open Information Security Foundation contribution agreement at
   https://suricata-ids.org/about/contribution-agreement/
6. Clone the Github repo & get started!
Want to know some tips and tricks?
Some tips and tricks to keep in mind

1. Start with ‘beginner’ tasks to familiarize with the codebase
Some tips and tricks to keep in mind

2. Gradually increase complexity as your confidence grows
Some tips and tricks to keep in mind

3. Read the descriptions, again and again

4. If nothing make sense, do a keyword search!
Some tips and tricks to keep in mind

5. Ask questions & clear your doubts  https://forum.suricata.io/
Some tips and tricks to keep in mind

6. Be mindful about commit messages, naming conventions!

- Wrap subject line at 50
- Wrap commit body at 72
- Consistent variable names
Some tips and tricks to keep in mind

1. Start with ‘beginner’ tasks to familiarize with the codebase
2. Gradually increase complexity as your confidence grows
3. Read the descriptions, again and again
4. If nothing makes sense, do a keyword search!
5. Ask questions & clear your doubts
6. Be mindful about commit messages, naming conventions!
7. Submit a Pull Request & ask for feedback
Interning with OISF through Outreachy
1. Initial application
   - Includes answering 4 essay questions

1. Make contributions
   - Pick 1 or 2 projects from the list of internship projects
   - Contact project mentor
   - Start contributing!

1. Final application
   - Record at least one contribution
   - Create a final application
Last but not the least,

- Be passionate about what you do
- Do NOT self-reject
- Believe in yourself
- Be confident!
Many thanks to the awesome team at OISF & Outreachy!

linkedin.com/in/tharushijayasekara/

twitter.com/tharushi96

tharushi96@gmail.com
THANK YOU!

Visit our YouTube channel for more about Suricata and our webinar events -
https://www.youtube.com/c/OISFSuricata

Connect with the community -> https://forum.suricata.io/