Modbus implementation already exists for Suricata but in order to make it more secure and trustworthy, we'd like to convert it to Rust.

Below is a (very) brief step by step overview of going about the implementation:

1. Get to know more about the protocol from the official RFC.
2. Match the RFC defined header structs and handling with C code and understand and note down the flow and handling.
3. Create basic header structures in Rust. (These do not have to be same as C but should follow the protocol standards (e.g. alignment and data types of certain fields) as per the RFC)
4. Use test data from C for headers and try to write a parser such that those tests pass as they did in C.
5. After the header parsers work in Rust, move to UDP implementation of packet handling (start with creating a `<Protocol>State struct to store all important info`).
6. Try to replicate UDP handling in C for the protocol and make unit tests pass for one transaction only.
7. Integrate with C, make sure the tests pass.
8. Find PCAPs for the application layer protocol over UDP covering all the modes (in case there are diff modes of operation for a protocol e.g. diff versions of protocol)
9. Write a (few) suricata-verify tests (you could use createst.py to create those for you with the PCAP) with those PCAPS and make sure all the tests pass.

This completes UDP conversion of the protocol and some basic chassis for protocol over TCP (structures and header parsing).

10. Try and replicate the TCP implementation in C of the protocol.
11. Learn more about stream handling and how Suricata does internal reassembly of packets, etc.
12. Repeat steps 6-9 for TCP.

This completes TCP conversion of the protocol as well but all works only for one transaction.

13. Try adding more transactions to tests and make them pass.
14. Make sure suricata-verify tests pass too.

This completes multi-transaction handling of the protocols.

15. Check out TCP stream gap handling for other protocols and try to implement the same for the protocol in Rust.
16. Enable gap handling for the protocol in C and integrate Rust code.
17. Find out or create PCAPs with TCP stream gaps, add tests with them to suricata-verify, make sure they pass.

This almost completes the conversion of the protocol.

18. Enable logging for the protocol, convert json logging from C to Rust.
19. Test that logging results pass with `--enable-debug-validation` and jsonbuilder does not panic anywhere.
20. Write suricata-verify tests for the same, make sure they pass.

This completes the protocol conversion to Rust. Now, fix bugs as when they arrive. :)

Pro tips:
- Take inspiration from other protocols, smaller/popular protocols (e.g. DNS, SIP) are easier to follow.
- Git is your best friend. (e.g. to see how a particular protocol's conversion progressed check all the closed PRs corresponding to that, a lot of useful content might be there)

Related issues:
Related to Task #3195: tracking: rustify all input New
Related to Task #2778: tracking: port app-layer parsers to Rust New
Related to Feature #2096: eve: event_type for MODBUS Closed
Has duplicate Optimization #2782: Convert Modbus from C to Rust Rejected

History
#1 - 09/18/2020 05:55 PM - Victor Julien
- Related to Task #3195: tracking: rustify all input added

#2 - 11/09/2020 05:16 AM - Shivani Bhardwaj
- Related to Task #2778: tracking: port app-layer parsers to Rust added

#3 - 11/09/2020 05:16 AM - Shivani Bhardwaj
- Related to Feature #2096: eve: event_type for MODBUS added

#4 - 11/09/2020 05:18 AM - Shivani Bhardwaj
- Related to Optimization #2782: Convert Modbus from C to Rust added

#5 - 12/08/2020 10:44 AM - Victor Julien
- Status changed from New to Assigned
- Assignee set to Juliana Fajardini Reichow
- Target version set to 7.0rc1

#6 - 12/08/2020 10:46 AM - Victor Julien
- Related to deleted (Optimization #2782: Convert Modbus from C to Rust)

#7 - 12/08/2020 10:46 AM - Victor Julien
- Has duplicate Optimization #2782: Convert Modbus from C to Rust added

#8 - 12/11/2020 04:23 PM - Juliana Fajardini Reichow
- Status changed from Assigned to In Progress

#9 - 12/23/2020 04:43 PM - Simon Dugas
Looks like there is some good progress on this ticket with https://github.com/OISF/suricata/pull/5675
We have also been working on an implementation and plan to have a PR ready in January. It is currently going through internal code review and qa. It features moving the app-layer, logging and detection to rust.
The modbus parser itself will be a separate rust crate that is part of a library of crates for security-aware protocol parsers.
Our apologies for not updating the ticket when we started development a couple months ago :).

#10 - 12/23/2020 05:07 PM - Victor Julien
Hi Simon, thanks for the update. Appreciate the work you are doing. At the same time, this is a bit tricky for us as Juliana is working on this as part of her internship with us for Outreachy. We'll have to look at how to handle that as I'd hate there to be duplication of efforts. Are there other things you plan to contrib that are not reflected by tickets yet?

#11 - 12/23/2020 08:14 PM - Simon Dugas
Hi Julien. Whichever way you decide to move forward is good with us.
There are a bunch of protocols we are looking to implement down the road but the ones we are actively working on are modbus (January 2021), diameter for LTE/4G (March 2021) and TCAP/MAP for SS7 (April 2021). I can't find redmine tickets for these at the moment.
For other protocols like ENIP and DNP3, we can notify you via redmine once they are assigned to a developer on our end to avoid duplication.

#12 - 12/24/2020 11:00 AM - Victor Julien
Thanks Simon. Could you create tickets for those plans?

#13 - 12/28/2020 03:29 PM - Juliana Fajardini Reichow
Assignee changed from Juliana Fajardini Reichow to Simon Dugas

Changing assignment to Simon Dugas, based on latest comments and agreements.

#14 - 01/04/2021 04:42 PM - Simon Dugas
Thank you, our pull request is coming soon for this.

The tickets for SS7 and diameter have been created:

- #4249
- #4250

#15 - 01/19/2021 09:19 PM - Simon Dugas
Pull request is open: https://github.com/OISF/suricata/pull/5750
You are also welcome to review our parsing library: https://github.com/CybercentreCanada/sawp

#16 - 03/11/2021 08:20 PM - Philippe Antoine
- Status changed from In Progress to In Review

#17 - 05/07/2021 03:13 PM - Simon Dugas
Pull request https://github.com/OISF/suricata/pull/6090 has been merged.

We are still finalizing lower priority fixes from the previous code review comments:
1. Fix for write multiple coils check identified in https://github.com/OISF/suricata/pull/5770
2. Adding a more detailed suricata-verify test to follow https://github.com/OISF/suricata-verify/pull/486

#18 - 05/10/2021 10:28 AM - Victor Julien
- Status changed from In Review to Closed

https://github.com/OISF/suricata/pull/6090