TDDE21 - HIPv2

Payam Tavakoli, Jens Lindgren, Tommy Johansson
Niklas Karlsson, Adrian Byström

December 2021

1 Introduction

1.1 Project purpose and goals

The purpose of this project has been to extend the Host Identity Protocol (HIP) implementation OpenHIP with support for OpenSSL 3.0.0 and NAT Traversal according to RFC 9028[1]. The project is a part of the course *TDDE21 Advanced Project: Secure Distributed and Embedded Systems* given at Linköping University

2 Background

2.1 Host Identity Protocol (HIP) and OpenHIP

The Host Identity Protocol is a protocol for identifying clients, services and devices in a cryptographically verified manner that is independent of the IP addresses used. It is designed to increase security by decoupling the identity of a host from its IP address and enforcing a layer of encryption while minimizing the attack surface for denial-of-service attacks. Commercial implementations of HIP already exists and are leveraged by companies such as Tempered to deliver security solutions for their client networks. OpenHIP is an open source implementation of HIP, and the code base that the project this report is based on was working with.

2.2 CORE

CORE stands for Common Open Research Emulator, it is developed by the Boeing Company. Even though CORE is an open source project it is copyrighted by the Boeing Company. However, said copyright is quite permissive and anyone can make modification to the source code as they see fit. CORE is a tool for emulating networks on one or more machines. The emulator has plethora of different API's for interfacing with it, and the one the team used was the Python API. This is because the Python API is what the OpenHIP project is using when testing its protocols. To write test the programmer simply imports and installs the necessary CORE libraries to the test script written in python, where they then can define nodes, prefixes and more needed objects for the emulated network. The test script is then fired up and can be manipulated either through a command line prompt or the GUI built into CORE. These emulated networks can work in isolation, but they can also be connected to live networks.

[1]https://datatracker.ietf.org/doc/rfc9028/

2.3 OpenSSL

OpenSSL[2] is an open source library and toolkit for general-purpose cryptography and secure communication over the Transport Layer Security Protocol (TLS) formerly known as the Secure Sockets Layer protocol (SSL). The TLS protocol is most prevantly used for securing network communication and is found in many different applications, from email to instant messaging. Its most visible usage however is securing HTTPS. All in all, it is used to secure by adding privacy and integrity for the connection between two endpoints. In the terms of HTTPS, two applications (the web server and the web browser)[3]. OpenSSL is developed and maintained by the OpenSSL team and is, as many other open source projects, run by a committee of maintainers and developers. This committee is called the OpenSSL management committee and it is the highest deciding organ within the OpenSSL project. The OpenSSL library and toolkit is available for all major operating systems, both for desktops and servers. There is different versions of the OpenSSL protocol, with the latest stable release being 3.0.0. There is also a LTS (long term support) version (1.1.1) that users of versions older than that are encouraged to upgrade to. These different versions differ a lot, with many of parts of the older API being deprecated in version 3.0.0[4]

2.4 NAT Traversal

Network address translation traversal (NAT Traversal) is a networking technique of maintaining internet connections across different gateways. This technique allows the network to assign private IP-address within it, but still only having a single outward facing IP-address to the larger Internet. With technique one can solve the problem of IP-address shortage, making away with the need to every single device needing its own outward facing IP-address to communicate with the Internet. This technique also hides devices within the network, protecting the communication channel from hackers.

3 Method

To achieve a more efficient workflow in this project the team split up in two parts, where one half of the group worked on the implementation of NAT traversal and the other half worked on the upgrading the protocol from OpenSSL 1.1.1 to OpenSSL 3.0.0. The entire team had a meeting at the start of every week where the current status of each sub-projects where discussed, and what goals each sub-project had until the next weekly meeting. If there were any issues pertaining to the work each sub-team had to do, the team would discuss this during these meetings as well.

[2]https://www.openssl.org/
[3]https://datatracker.ietf.org/doc/html/rfc5246#section-1
[4]https://www.openssl.org/docs/man3.0/man7/migration_guide.html

At the end of the project the sub-teams finalized their respective sub-projects to then write this project report.

4 Results

4.1 Before half time presentation

4.1.1 Getting started

Initially the team set up communication and administrative resources such as a Drive folder, Teams room and an Overleaf document for the report so that the members could easily contact each other and share resources and files while the university largely operated in distance mode. When starting development the team was given access to the Bitbucket containing OpenHIP by the examiner for the course that this project is a part of. Permission was granted to create branches in the repository, so a fork of the project was created to be able to version control the teams development progress.

4.1.2 Running OpenHIP and Core

The first challenge was setting up the development environment around Core and OpenHIP. This was the first major roadblock the team encountered, and took a significant amount of time to deal with for three reasons. Firstly, the team initially had next to no experience with either project, so the concept of what a correct setup looked like was not known to the team. Secondly, the provided instructions from the course information were outdated and incorrect, and did not lead to a functioning configuration. Finally, the CORE project was dealing with a bug in its dependencies that caused its build process to fail.

Once a working configuration had been identified and the team managed to get it to run, we wrote a new instruction guide for how to install and run Core and OpenHIP on a virtual machine. This was done so that all team members had the same base configuration to work from. This instruction guide will be provided in this report with the hope that future teams developing OpenHIP will not encounter the same issues we did.

Setting up the development environment took considerable time, in an attempt to not waste time, the team also read up on documentation for CORE and OpenHIP, as well as the HIPv2 RFC and its extension RFCs.

4.1.3 Implementation of OpenSSL 3.0

OpenHIP is currently implemented using OpenSSL 1.0.0. Some efforts were made previously to upgrade to OpenSSL 1.1.0 when it was released, but as far as we can tell that implementation never reached a working state and was never merged to the primary development branch for OpenHIP. OpenSSL 3.0.0 was available as a beta release at the beginning of the course and officially released in early September of 2021. Due to the 1.1.0 implementation being incomplete, in an unknown state and targeting a different version than the desired 3.0.0 the sub-team chose to start the implementation from the main branch. The goal for this sub-team was to implement support for OpenSSL 3.0.0 while preserving backward compatibility for OpenSSL 1.1.0.

4.1.4 Implementation of NAT Traversal

OpenHIP currently employs Legacy ICE for NAT Traversal, ICE stands for Interactive Connectivity Establishment and Legacy ICE uses Session Traversal Utilities for NAT (STUN) for this purpose. We are to implement Native ICE to give HIPv2 NAT Traversal support. Native ICE replaces STUN with HIP UPDATE packets to simplify demultiplexing of different protocols.

Legacy ICE used Rendezvous Relay Servers (RVS) to help two hosts to establish a direct connection between them by relaying HIP packets. These servers are replaced by Control Relay Servers (CRS) and Data Relay Servers (DRS) in Native ICE, separating the data and control plane HIP packets. All HIP packets over the control and data plane are encapsulated by UDP using the CRS and DRS.

4.2 Last half

After the half-time presentation in the course the team continued working on the project with the plan presented in said presentation. The team was still split into two different groups, one with focus on implementing OpenSSL 3.0 and another focusing on implementing NAT-Traversal.

4.2.1 Implementation of OpenSSL 3.0

The re-implementation of OpenHIP using OpenSSL 3.0.0 started by setting up a virtual machine with OpenSSL 3.0.0 as the default OpenSSL version and attempting to compile the OpenHIP project. This turned out not to be possible due to the significant differences between the 1.0.0 and 3.0.0 versions of OpenSSL. Since OpenHIP is constituted largely of calls to the OpenSSL library, has high code cohesion as well as significant code duplication. Large changes were required to translate the existing OpenHIP implementation to an equivalent version using OpenSSL 3.0.0. The primary goal became to perform a full translation of deprecated function calls and data structures according to the OpenSSL documentation. After that the plan was to begin testing and debugging against the existing test suite as well as adding additional tests to increase code coverage. The translation work took longer than anticipated due to reasons stated previously, and while the sub-team were able to complete a compilable version of OpenHIP with it using OpenSSL 3.0.0, the sub-team were not able to test the implementation due to time constraints. It is the hope of the team that the task of debugging the new implementation can be passed on to a future team.

4.2.2 Implementation of NAT Traversal

During the last half the focus was on getting the CRS working since it is the most vital part for NAT Traversal. This was because it helps establish a connection between the initiator and responder through the base exchange (BEX) where both or only one of them are behind a NAT. Both team members in this subgroup focused on implementing CRS together to reduce the chance of missing some specifications and due to the complexity of NAT Traversal.

The first step was to implement the changes to HIP packets, header values and datastructures according to RFC 9028 for relay information of HIP associations used by the CRS. However, some old header values collided with the new ones and some header values that should already exist according to RFC 5770 and RFC 8445 did not. This was fixed by adding the missing values and commenting out the old values which collided with the new.

It is the hope of the team that the work of implementing DRS and HIP UPDATE keep-alives as well as writing CORE tests for NAT Traversal can be passed along to a future team.

5 Discussion

5.1 OpenSSL 3.0

In the end we managed to get the application to compile with OpenSSL 3.0, however it does not pass the tests that existed for the OpenSSL 1.0 version, which is not very strange at all, since the program has gone through a multitude of changes due to the vast differences in OpenSSL 1.0 and OpenSSL 3.0. Porting it over from OpenSSL 1.0 to 3.0 was a much bigger task than anticipated. OpenSSL has gone through a major overhaul with the update to 3.0 and therefore we had to rebuild many parts of OpenHIP. 1500 line insertions, 600 line deletions, changes in 16 files shows that it was no small task to just get it to compile. We were working in the dark, but the program is now in a state where it compiles, which means there is a chance to see what is going on in the program and to understand where it goes wrong. One of the biggest hurdles when upgrading to OpenSSL 3.0 was the highly coupled nature of the source code for the OpenHIP protocol. This together with the fact that the deprecation warnings from compilation could not be "skipped", and stopped compiling whenever the compiler hit one of these deprecated functions within the code made it near impossible to potential split the workload even further. Had the team been able to actually split the workload amongst themselves into even smaller parts, then maybe more progress could have been made. But as it stands now, what the team leaves as future work is to test that the protocol actually works as intended. It is the hope of the team that this report will be of help to them, so that the preceding team does not have to start with a similar lack of knowledge of both the protocol and the OpenSSL documentation as this team did.

5.2 NAT Traversal

Due to highly coupled coupled code, as previously mentioned, and the enormous sizes of functions it was not an easy task to identify where to edit the code to add the CRS functionality. The fact that the new header values used for Native ICE overrides some existing header values that was not associated with NAT Traversal was quite confusing. Missing header values made it even more confusing. But when this was remedied by adding and changing header values and the mysteries of how to make OpenHIP use CRS the implementation started to pick up some speed.

The work load could have been divided into to implement CRS and DRS in parallel, but this would probably had led major merge conflicts and more buggy code due to the complexity of the task at hand. Same goes for dividing implementation and creating CORE test for NAT Traversal, this could probably lead to tests that do not test every aspect of NAT Traversal and risk that the implementation of CRS would be incomplete. Now there exists a probably functioning CRS in OpenHIP, only lacking DRS and HIP UPDATE keep-alives for OpenHIP to fully migrate to Native ICE from Legacy ICE.

6 Future Work

6.1 Packet Verification Tool

As development on OpenHIP progresses it would be valuable to create a tool specifically for inspecting the packets sent during the BEX and subsequent communication. Independent correctness could be analyzed more easily by comparing sent packets with RFC specifications. Motivation for this tool comes from a case where prior project work suffered delays due to not being able to easily identify a failed checksum during the BEX. The tool could act as a good complement to automatic testing by enabling developers to reason about the desired and actual OpenHIP functionality. It would also provide another means of introducing the OpenHIP protocol to new developers and so hopefully drive development overall. Finally, the tool could be used, once correctness of some implementation was confirmed, to perform a ”base case” communication sequence that could be used to verify correctness between implementation changes.

6.2 Extension of Automatic Tests

During the implementation of OpenSSL 3.0.0 it seemed that the code for various HIP cipher methods was highly suspect, and we took special note to attempt to write more test cases for the various ciphers OpenHIP is supposed to support. It seems to us that there is reason to think that some of the ciphers that supposedly already have support implemented do not work as intended. It would be good to make a concerted effort to increase test coverage of the various cipher types used, as well as the BEX cipher decision cases.

6.3 CORE tests for NAT Traversal

As of now there exists no test for NAT Traversal in OpenHIP which would be helpful to assess if the NAT Traversal implementation is correct. This should test that the BEX is completed successfully, CRS successfully relays I1, R1, I2, R2 and UPDATE packets, and DRS successfully relays packets in the data plane. It could also be of use to create tests that verify the use of non-critical parameters sent in CRS, including test cases where the parameters are dropped.

7 Advice

7.1 Advice for OpenSSL

A good first step for OpenSSL would be to implement the Packet Verification Tool we describe in Future Work 6.1.

7.1.1 Guide for running CORE & HIP

Run following commands in the command line of a virtual machine running Ubuntu 18.04

```
sudo apt-get install -y gawk autoconf automake libtk-img openssh-client \
openssh-server ebtables git libssl-dev libxml2-dev
sudo update-alternatives --set ebtables /usr/sbin/ebtables-legacy
git clone https://github.com/coreemu/core.git
cd core
./bootstrap.sh
sudo ./install.sh -l

# OPENHIP
cd ..
git clone <eran bitbucket för openhip>
git checkout hipv2
./bootstrap.sh
./configure
make
sudo make install
cd test
sudo python3 test_switch.py
```

7.2 Advice for NAT Traversal

7.2.1 Similarities Between CRS/DRS vs. RVS

The way OpenHIP deduces if the device running OpenHIP is an ordinary device or a type of relay server is very similar. Take inspiration from how the

RVS from the Legacy ICE implementation and the CRS from the Native ICE implementation does this when implementing the DRS for Native ICE.

7.2.2 Usage of non-critical parameters in CRS (Additional)

Right now there are non-critical parameters such as transaction pacing and NAT traversal mode parameters. These are currently sent in the BEX only to be dropped. This was done since RFC 9028 did not specify how the parameters were to be used after the BEX, only that they were non-critical and therefore dropping them is a valid option in the exchange. More specifically a point in time should be specified for when the transaction pacing calculation should occur. The calculation itself should also be decided and then, the option to enforce the parameter should exist.

For the NAT traversal mode parameter, there should exist a way to decide the priority order for each server, and the usage of that instead of the default list in R1 and I2. Furthermore the option chosen from each list should be able to be enforced in each client as a part of the exchange. This requires an additional option in the setup.