

TDDE21 - Project Report

Niklas Granberg, Fabian Haugen, Christoffer Lindström, Eric Nylander, Anna Pestrea

December 16, 2019

1 Introduction

1.1 Background about the project

This report contains the results of the project made in the course *TDDE21: Advanced Project: Secure Distributed and Embedded Systems*. The project consist of extending a security protocol called HIP or Host Identity Protocol (RFC7401¹) made by the IETF. The extensions of HIP concern Multihoming (RFC8047²) and Mobility (RFC8046³). The project was made by a group of five students from Linköping Univeristy. For the project an already existing version of HIP was provided together with the report from last years students.

1.2 HIP - Host Identity Protocol

The Host Identity Protocol allows hosts to separate the identifier and locator functions of the IP address. This means that the hosts can “securely establish and maintain (a) shared IP-layer state” (RFC7401⁴). HIP is also based on the Diffie-Hellman key exchange, so public keys are used in the peer authentication.

This protocol is meant to make the security between two hosts higher. One example of this is that HIP is very resistent towards denial-of-service (DoS) attacks and man-in-the-middle-attacks. It can also be used together with other security protocols such as Encapsulation Security Protocol (ESP), in order to get even higher security.

2 Method

We started by having “labsessions” together, where we sat down and programmed. Firstly we looked through already existing implementations that was provided to us. After doing this we realised that there were some things that weren’t functioning as intended, so a list of all things that had to be implemented or fixed was made. We then split the work so that each student sat down with a task from the list and worked on it. We also discussed and got help from each other, as some parts affected something other people were doing. The online tool Trello was used to specify weekly tasks⁵. The detailed plan can be viewed down below. The names in cursive is the names of the people working with the subject.

¹<https://www.rfc-editor.org/rfc/rfc7401.txt>

²<https://www.rfc-editor.org/rfc/rfc8047.txt>

³<https://www.rfc-editor.org/rfc/rfc8046.txt>

⁴<https://www.rfc-editor.org/rfc/rfc7401.txt>

⁵<https://trello.com/>

1. From previous report: Fix HIT suite negotiation plus I2 receival and keymat generation, which is related to the HIT suite (*Niklas and Fabian*)
 - Check exactly how the process of the HIT suite selection happens
 - Check which HIT suite we use for each specific HIT
 - Implement code so that the correct HIT suite is chosen after the list we have implemented
 - Use SSL HKDF for keymat generation in `hip_keymat.c`
 - Tests
2. From previous report: Fix ICMP (*Anna and Eric*)
 - Handle eventual problems that will the test might give
 - Create tests
3. From previous report: Fix OpenSSL 1.1.1 (*Christoffer*)
 - Fix so that HIP can be run with OpenSSL 1.1.X
4. Implement mobility RFC8046 (*everyone*)
 - Create Credit-based Authorization (CBA) for the HIT that is stored
 - The LOCATOR parameter in [RFC5206] has been renamed to LOCATOR_SET. Does not have any actual effect on the program itself.
 - Deprecate old address if the time limit is exceeded while Base exchange is run on the new IP address or if we have changed to a new IP address
 - Same thing as before, but do the HIP re-keying different. It should be unverified until it's activated or deprecated.
 - Send HOST_ID in an update packet and handle on receival
 - a host may add the source IP address of a received HIP packet as a candidate locator for the peer even if it is not listed in the peer's LOCATOR_SET, but that it should prefer locators explicitly listed in the LOCATOR_SET.
 - Avoid processing locator sets that have recently already been processed.
 - We will not do the rendezvous extension that is specified in RFC8046 as it requires rfc 8004
5. Implement multihoming RFC8047 (*everyone*)
 - Implement so that we can add a new address or make sure that it is already implemented (this part is needed for mobility and might be done when mobility is implemented)
 - Send all information about all LOCATORS:s that we want to continue using
 - Check if the old SPI is the same as it was before
 - Set up a new SA (and change the old one) or set up new SA and keep old one

3 Results

3.1 Before half-time presentation

This section contains information about what kind of problems we had before the half-time presentation.

3.1.1 Make previous version work: Implementing ICMP

At first we tested to implement a solution under the recommended code part (the if case containing `HIP_PROTO_VER` in `src/protocol/hip_input.c`). Then we tried to create a test to test our code. The test was to be created in the file `src/util/script/hiptest.py`. However we encountered some problems here. We cannot get the HIP version for the different nodes from this part of the program. We tried multiple times but realised that we had to give up as we cannot do a test on something that does not exist at this point.

3.1.2 Main challenges

Some of the main challenges was setting up the environment, building the project, making sure that old code works, understanding the code and making the OpenSSL upgrade and the tests. We did come across many problems and sometimes it was hard to troubleshoot exactly what was wrong. It should also be noted that many of us used different systems and so forth we could have different problems depending on the system that was used. We all installed Virtual Machines (as not all of us had Ubuntu) or used SSH to connect to a computer with Ubuntu. This of course lead to many irritating smaller problems that sometimes was limited to a single person, meaning that the rest of the group was unable to help.

3.2 Half-time presentation

Here we present how far we had come by the time of the half-time presentation and how we planned to proceed from this point.

3.2.1 Current work so far

In the half-time presentation we presented our current work so far. By this time we had finished some of the work, which included:

- Updated so that HIP could be run with OpenSSL 1.1.0 instead of OpenSSL 1.0.2
- Testing the functionality of the protocol using the CORE emulator - tests existed for base exchange, mobility, and rekeying
- Implemented code in `src/protocol/hip_input.c` for HIP version mismatch. Tried to test with the file `src/util/scripts/hiptest.py`, which was unsuccessful.
- Implementing client HIT suites selection
- Keymat generation is implemented except for Declining based on Diffie-Hellman group which should be possible

3.2.2 Plan for the second half

By this time we had come halfway through the project and not yet implemented what we from the beginning set out to do. Because of this we felt that we needed a new plan for the last half of the project. The plan for the second half of the project time was to:

- Test HIP version mismatch handling between hosts (with Wireshark)
- Implement and test: HIT suite negotiation
- Implement and test: Keymat generation
- Implement test for Multihoming
- Update Mobility for HIPv2
- Implement Multihoming

3.3 End results

The sections below contain information about what was done after the half-time presentation. After the half-time presentation we decided that we don't care about backwards-compatibility, it was mentioned in RFC7401 (HIPv2)⁶.

3.3.1 HIT suite negotiations

Figure 3 shows a summary of a Base EXchange (BEX). The parts marked in red are the areas where much of the work can be seen and is also where most of the problems still exist.

The report from last year presented three different suggestions to solve the problem regarding the HIT suite negotiation. After looking through the suggestions and a idea from one of the main developers of openhip, we realised that the solution is that each HIT contains the selected HIT suite. Therefore we could select the HIT suite and then easily make sure that both the Initiator and Responder knows the selected HIT suite. A lot of hard coded HIT suite list implementations were also removed.

There were more problems with HIT suite negotiation. The previously implemented HIT suite negotiation did not cover all of the requirements stated by the RFC. When the initiator receives an R1 packet, it needs to first compare its current HIT against the list. If that fails, compare the HIT suite list with its own HIT suite list and then restart the BEX with a new, compatible HIT. If that fails the BEX is quietly aborted. These have been partially implemented, with the BEX restarting still missing.

One major problem that still exists within the HIT suite negotiation is the default values. In many cases when a HIT suite is received and it is not 1, 2 or 3, it is assumed that it should be HIT suite 1. This can lead to behaviours where the negotiation should fail, but instead continues. When these negotiations occur, it makes error searching extra hard. This error occurs in many places in the code base and can be hard to track down.

At the end of the work period, there is another major problem still existing in the code. The complete HIT suite negotiation does not work. There is a problem where the suite that is agreed upon when parsing the R1 packet, is not the correct one. The problem appears during the tests

⁶<https://www.rfc-editor.org/rfc/rfc7401.txt>

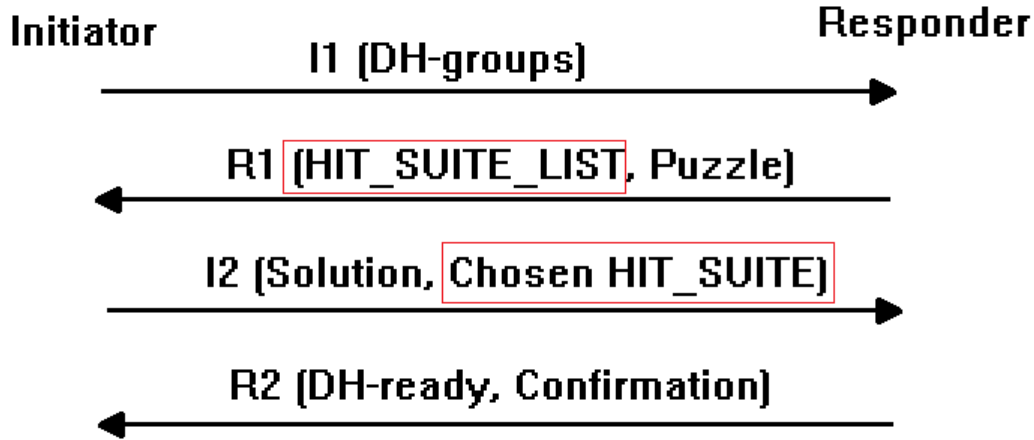


Figure 1: A summary of a BEX with changed parts in red

and the cause of the suite mismatch is unknown. What appears to happen is that both nodes in the test read the same HIT suite from one of the nodes configuration file.

2001:21:6a99:904e:61ae:1d0f:9ad5:9e20

Figure 2: An example of a HIT. Note that in the second group the initial two zeroes are not written. The underlined number is the hit suite id

3.3.2 HIP version mismatch

The intended course of action on a HIP version mismatch is that the host should send an ICMP “Parameter Problem” packet. The issue in solving this is that the `hip_handle_packet` function is located in the `hip_input.c` file. Errors relating to the content of the packet are handled in `hip_main`, however. There is an implementation of sending an ICMP Parameter Problem packet in the usermode code that handles esp information related to IPsec.

As of right now, the branch version `_mismatch` contains a fix that sends a packet on version mismatch with the correct information set in the packet. However, this has an illogical dependency tree where the main loop of the hip protocol contains some functionality that it should not. The general implementation of the `send_icmp` function should probably be moved to the `hip_util.c` file, but time constraints limited our ability to verify whether or not this was the best location for it.

Testing the actual implementation of the fix requires the use of two machines that are running HIP since the HIP protocol version is defined as a constant in the `hip_proto.c` file in the include directory. Thus, running two different versions of HIP requires two different installs of the protocol in communication.

3.3.3 Testing at packet level

The tests that are currently implemented in the openhip repository are abstract and simply read from the log file generated by the protocol being run on two HIP nodes. These look for messages

such as “Ping received” and “I2 packet received” in this log file and thus have no notion of the content of packets sent between the hosts.

One type of test that would be valuable in testing the protocol would include the ability to read, edit and replay packets between hosts. We were unable to implement this type of test due to the interception of packets at the transport layer being a difficult problem to tackle in the allotted time. This type of test would be useful in the testing of values to the `hip_handle_packet` function in the `hip_input.c` file.

A proposed alternative testing method is instead setting up a unit-test suite for relevant methods in future development. For instance, having a proper set of stubs for the testing of the aforementioned function would be useful in developing a test for the HIP version mismatch implementation.

3.3.4 Implement Mobility and Multihoming

We first looked at implementing CBA (Credit-based authentication) since it was not yet used in the code for the previous version, but was labeled with MUST in the rfc for the new version. It was decided that implementation of CBA was not to be prioritized and the work so far is on the `cba` branch.

We read through rfc 8046(Mobility) and matched it to the implemented code. We implemented that `Host_ID` are sent with the update packet and received and processed. Currently it always sends it even though rfc states may and checking the `host_id` with ECDSA is not implemented.

The rfc states that reprocessing equivalent locator sets should be avoided. This has now been implemented.

CBA (and the new credit heuristic specified in the rfc) was not implemented as Gurtov did not see it as a high priority and we were running out of time.

Specification text regarding the mobility event in which the host briefly has an active new locator and old locator at the same time (a "make-before-break" mobility scenario) has been added. We believe that this is already implemented as break-before-make leads to the same consequences. However make a double-check on this

a host may add the source IP address of a received HIP packet as a candidate locator for the peer even if it is not listed in the peer's `LOCATOR_SET` but it should prefer locators explicitly listed in the `LOCATOR_SET`. This was implemented.

On Gurtov's recommendation we did not set much time on Multihoming.

4 Discussion

We used Trello⁷ (a online tool that is similar to a scrum board) a lot in the beginning, but then we used it less and less frequently. After we stopped using it, we faced more problems and also became less productive compared to before. We could have continued using trello, even though we didn't have many tasks to put up at it. At the end, we did use trello a little bit and our work became more structured.

⁷www.trello.com

5 Future Work

If someone in the future want to extend HIP then it might be a good idea to check than you have access to an fully functioning HIP implementation before starting to extend it. A lot of our time was consumed to fixing fault that was found in the given implementation.

5.1 To do for future works

This is still not implemented regarding HIT suites and their negotiation and comes from the RFC⁸:

- If the received Responder's HIT in the I1 is NULL, the Responder selects a HIT with the same HIT Suite as the Initiator's HIT. If this HIT Suite is not supported by the Responder, it SHOULD select a REQUIRED HIT Suite from Section 5.2.10, which is currently RSA/DSA/SHA-256. Other than that, selecting the HIT is a local policy matter.
- The system MUST check that the Initiator's HIT Suite is contained in the HIT_SUITE_LIST parameter in the R1 packet (i.e., the Initiator's HIT Suite is supported by the Responder). If the HIT Suite is supported by the Responder, the system proceeds normally. Otherwise, the system MAY stay in state I1-SENT and restart the BEX by sending a new I1 packet with an Initiator HIT that is supported by the Responder and hence is contained in the HIT_SUITE_LIST in the R1 packet. The system MAY abort the BEX if no suitable source HIT is available.
- The hit suite implementation is questionable on certain points and needs to be redone. Right now it accepts hit suites with value 0 by assuming it should be value 1. Since 0 is a reserved value, it should not be accepted. This occurs in hitgen when first generating a HIT and during the base exchange when verifying HITs. It can also cause base exchanges to fail because of hit suite mismatches that should not happen.
- The HIT suite list length is currently too short. The list can keep 15 different suites in it, but is limited to 4 suites right now. This could need changing for future functionality.
- The HIT suite negotiation is currently not complete. The second step where it compares entire HIT_SUITE_LISTs, it does not restart the BEX with a new HIT at the senders side. It currently only stops the BEX.
- The HIT suite negotiation does not work. The negotiation does not conclude with the correct suite. The problem appears to be that both nodes in the test, read the same suites from a one of the nodes configuration files.

Then there are some more general parts of Openhip that are not implemented:

- It would be good to test if the ICMP packet is sent correctly when a HIP version mismatch occur. As of now, there is no way of testing our implementation.
- Mobility Implement that the public key match when HOST_ID using ECDSA is sent in the update paramater.
- You maybe want to move the sending of HOST_ID from send_update to send_update_with_locators
- Implement CBA. It was require in hipv1 already but and is still not implemented in hipv2. Check rfc 8046(mobility)

⁸<https://tools.ietf.org/html/rfc7401>

- Double check if make-before-break is implemented. We believed it to be so.
- Memory leaks. We noticed early on there are a fair bit of memory leaks but we forgot about it while trying to get the program to run. Maybe make this a point for next year.
- Multihoming

6 How to get Started

The code created from this project is available at the website bitbucket under the name “openhip”. It’s specifically located at the “hipv2”, “version_mismatch”, “cba” and “hipv2_suitefix” branches. The only thing that is implemented in the “version_mismatch” is the HIP version mismatch handling, while everything else that was done is on the other branch “hipv2”.

The code base is about 37 000 lines of code, so make sure to start at the right place. To understand an implemented subject, look at the corresponding file in the table below:

| Subject | Relevant files |
|-----------------------|--------------------------------------|
| HIP suite negotiation | src/protocol/hip_input.c |
| HIP version mismatch | src/protocol/hip_input.c, hip_main.c |
| HIT suite list | src/util/hitgen.c, hip_xml.c |
| CBA | src/protocol/hip_output.c |
| misc | src/include/hip/hip_types.h |

To run HIP we recommend using a computer with ubuntu 18.04. If you do not have a computer with ubuntu, then using a virtual machine (VM) is recommended. You also need to install OpenSSL (version 1.1.0), CORE and other minor installations that might be needed (for example a VM will from the beginning not have the proper commands that are needed to run the file configure.ac). If you want to test HIP there exist tests at src/util/scripts in the project.

When you have installed CORE there are two packets that you need to install that aren’t in the 18.04 ubuntu OS. These packages need to be installed manually: IpRoute1 (IpRoute2 exist but won’t be found) and libreadline6 (which is installed by downloading the deb-file for the 16.04 version, run the command “dpkg -i [file name]”). Otherwise follow Andrei Gurtov’s introductory mail or if you are lucky the branch for building on 1804 might work.

6.1 Testing

In the openhip repository exist a series of tests in openhip/src/util/scripts. In order to run these tests, a package named tuncctl needs to be installed. This exists in the apt repository under the package uml-utilities. This can be installed using:

```
sudo apt-get install uml-utilities
```

6.1.1 Running tests

In order to then run the tests, there is a Make script in the main folder of the repository. Make sure you are in the src folder when you run these tests. In order to run this, use the command:


```
sudo make check
```

Root privileges are needed to run the tests because edits are made directly in the pycore package, which needs elevated privileges, like the install script.

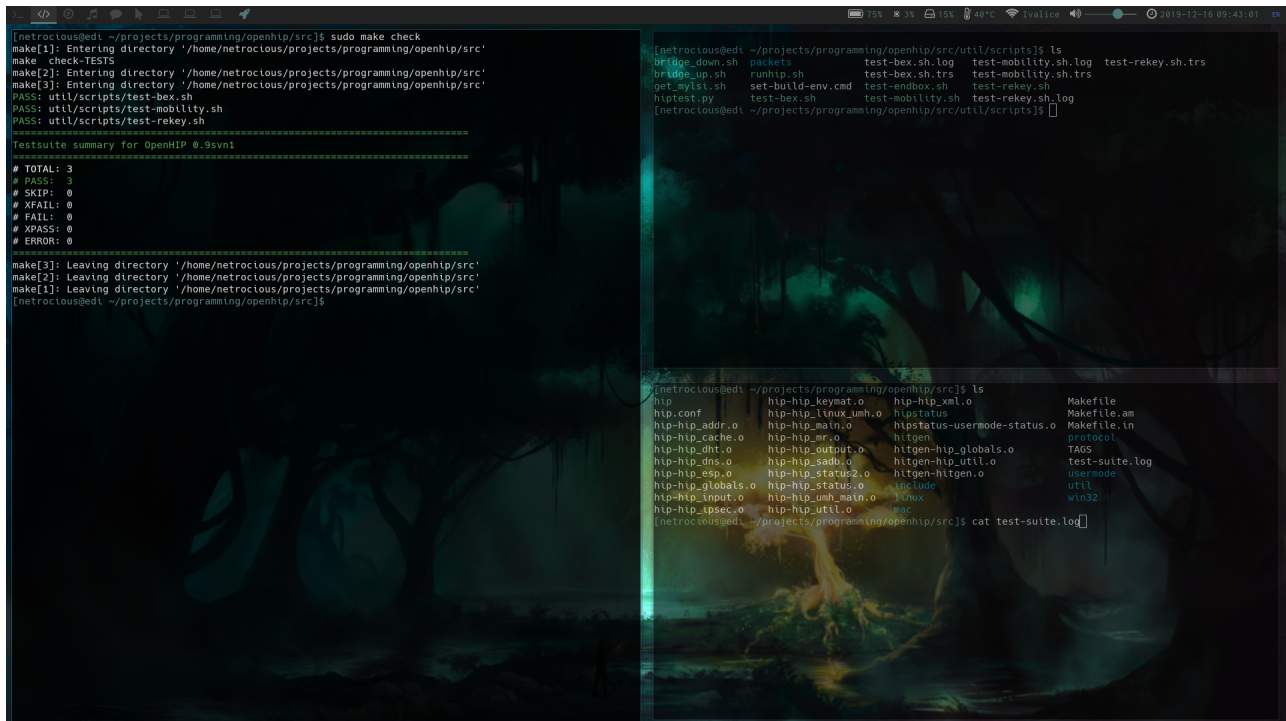
6.1.2 The tests

The test environment in which the tests are run is a setup with two nodes running HIP with a router connecting them. This allows for testing with ping and adding legacy nodes for testing connection to nodes not running HIP.

It is important to note that the test will use files if they already exists. Notably this is the conf, my_host_identities and known_host_identities. So if you alter code that should alter these files then delete .hiptest. This folder should be hidden in your home map. Remember to use sudo.

The tests that are currently implemented test a normal HIP base exchange, mobility and a host rekey. Each of these tests can be run individually by running the test file from the src directory with the related argument name. For example, to run a base exchange:

```
sudo util/scripts/hiptest.py bex
```



```
[netrocloud@edl ~/projects/programming/openhip/src]$ sudo make check
make[1]: Entering directory '/home/netrocloud/projects/programming/openhip/src'
make check-TESTS
make[2]: Entering directory '/home/netrocloud/projects/programming/openhip/src'
make[3]: Entering directory '/home/netrocloud/projects/programming/openhip/src'
PASS: util/scripts/test-bex.sh
PASS: util/scripts/test-mobility.sh
PASS: util/scripts/test-rekey.sh
=====
Test suite summary for OpenHIP 0.9v11
=====
# TOTAL: 3
# PASS: 3
# SKIP: 0
# XFAIL: 0
# FAIL: 0
# XPASS: 0
# ERROR: 0
=====
make[3]: Leaving directory '/home/netrocloud/projects/programming/openhip/src'
make[2]: Leaving directory '/home/netrocloud/projects/programming/openhip/src'
make[1]: Leaving directory '/home/netrocloud/projects/programming/openhip/src'
[netrocloud@edl ~/projects/programming/openhip/src]$
```

```
[netrocloud@edl ~/projects/programming/openhip/src/util/scripts]$ ls
bridge_down.sh  packets          test-bex.sh.log  test-mobility.sh.log  test-rekey.sh.trs
bridge_up.sh    runhip.sh        test-bex.sh.trs  test-mobility.sh.trs
get_mylist.sh   set-build-env.cmd  test-endbox.sh   test-rekey.sh
hiptest.py      test-bex.sh       test-mobility.sh  test-rekey.sh.log
[netrocloud@edl ~/projects/programming/openhip/src/util/scripts]$
```

```
[netrocloud@edl ~/projects/programming/openhip/src]$ ls
hip             hip-hip_keymat.o  hip-hip_val.o    Makefile
hip.conf        hip-hip_linux_umh.o  hipstatus        Makefile.am
hip-hip_addr.o  hip-hip_main.o     hipstatus-usermode-status.o  Makefile.in
hip-hip_cache.o  hip-hip_mr.o       hitgen           protocol
hip-hip_dht.o    hip-hip_output.o   hitgen-hip_globals.o  TAGS
hip-hip_dns.o    hip-hip_sadb.o     hitgen-hip_util.o  test-suite.log
hip-hip_esp.o    hip-hip_status2.o  hitgen-hitgen.o    usermode
hip-hip_globals.o  hip-hip_status.o  include          util
hip-hip_input.o   hip-hip_umh_main.o  linux            win32
hip-hip_ipsec.o   hip-hip_util.o     src
[netrocloud@edl ~/projects/programming/openhip/src]$ cat test-suite.log
```

Figure 3: Executing in the test environment

6.1.3 Testing directly

It is possible to run a session as set up in the tests by running the script

```
sudo make shell
```

This will open up two xterm windows connected to each of the HIP nodes. From here, it is possible to read log files and have the nodes communicate.

6.2 Errata

In section 2.2 of the document *Running OpenHIP under CORE* there exist a misspelling that confused us. The error resides in the command below:

```
mkdir \${SESSION}\_DIR/n2.conf/usr.local usr.local.etc.hip
```

This command will create a dictionary named *usr.local.usr.local.etc.hip*. However later commands that will be run are going to look into the folder *usr.local.etc.hip* instead of *usr.local.usr.local.etc.hip*. Change the command to the following command:

```
mkdir \${SESSION}\_DIR/n2.conf/usr.local.etc.hip
```

Then the correct behaviour will follow.