Deliverable D3.4
5G-PPP Security Enablers Documentation (v1.0)
Enabler Bootstrapping Trust

Project name	5G Enablers for Network and System Security and Resilience	
Short name	5G-ENSURE	
Grant agreement	671562	
Call	H2020-ICT-2014-2	
Delivery date	30.09.2016	
Dissemination Level:	Public	
Lead beneficiary	NEC	Felix Klaedtke, felix.klaedtke@neclab.eu
Authors	SICS Swedish ICT: Nicolae Paladi	

Foreword

5G-ENSURE belongs to the first group of EU-funded projects which collaboratively develop 5G under the umbrella of the 5G Infrastructure Public Private Partnership (5G-PPP) in the Horizon 2020 Programme. The overall goal of 5G-ENSURE is to deliver strategic impact across technology and business enablement, standardisation and vision for a secure, resilient and viable 5G network. The project covers research & innovation - from technical solutions (5G security architecture and testbed with 5G security enablers) to market validation and stakeholders engagement - spanning various application domains.

Disclaimer

Copyright notice

Contents

1 Introduction 4
2 Installation and Administration Guide 4
2.1 System Requirements 5
2.2 Enabler Installation and Setup 6
2.2.1 OS-level Configuration 6
2.2.2 Emulation environment configuration 7
2.2.3 SDN configuration 7
2.3 Troubleshooting 8
3 User and Programmer Guide 8
3.1 User Guide 9
3.1.1 Overview 9
3.1.2 Enabler Output 10
3.1.3 Open vSwitch enrollment 10
3.2 Programmer Guide 10
4 Unit Tests 10
4.1 Information about Tests 10
4.2 Unit Test 2252 11
4.3 Unit Test 2253 12
5 Abbreviations 12
6 References 13

1 Introduction

The physical infrastructure supporting 5G networks will be multiplexed among multiple tenants with own environments, similar to a cloud infrastructure model. In this model, SDN allows tenants to configure complex topologies with rich network functionality, managed by a network controller. The availability of a global view of the data plane enables advanced controller capabilities – from pre-calculating optimized traffic routing, to managing software applications that replace hardware middleboxes. However, these capabilities also turn the controller into a valuable attack target: once compromised, it can provide the adversary with complete control over the network [2]. Furthermore, the global view itself is security sensitive: an adversary capable of impersonating network components – such as virtual switches – may distort the controller's view of the data plane and influence the network-wide routing policies. Virtual switches are security sensitive elements in SDN deployments. They run on commodity operating systems (OS) and are often assigned the same trust level and privileges as physical switches – specialized, hardware components with compact embedded software. Commodity OS with large code bases are likely to contain multiple security flaws which can be exploited to compromise virtual switches. For example, their configuration can be modified to disobey the protocol, breach network isolation and reroute traffic to a malicious destination or hijack other network edge elements through lateral attacks. Such risks are accentuated by the extensive control a cloud provider has over the compute, storage and networking infrastructure – this requires long-term, absolute trust from the tenants in the integrity of the cloud provider's security perimeter.

The aim of this enabler is to attest the integrity virtual switch components and prevent virtual switch impersonation attacks by enrolling into the topology *only* attested switches, as well as storing the authentication keys in a trusted execution environment. This is expressed in two use cases described in [2], namely *attest integrity of a virtual switch in the SDN deployment* and *enroll attested virtual switch into SDN deployment*.

The final version of this enabler is due to be delivered in Release 2 -- including a comprehensive implementation of *integrity attestation of virtual network components* -- according to the description in Deliverable 3.1 [1] and Deliverable 3.2 [2]. The current manual addresses the functionality implemented in Release 1, namely, prototypes of *enrollment of integrity attested virtual switches* and *provisioning of switch-specific credentials to trusted execution environments (TEEs)*.

2 Installation and Administration Guide

Release 1 of the Bootstrapping Trust enabler consists of a set of libraries and configuration scripts that enable basic integrity verification and enrollment functionality. Given the reliance on software emulation (to emulate the functionality of TEEs, it is essential to note that the current release of the enabler *does not* provide any security guarantees.

The current enabler release contains the following elements:

1. A binary of the Open vSwitch containing modification in the module <lib/stream-ssl.c>
2. Implementation of the TEE for provisioning of switch-specific credentials (further referred to as "OVS TEE").
3. Implementation of the TEE remote attestation functionality, as an extension of the Ryu controller.
4. Installation and configuration scripts (further referred to as "Controllerapp").

Figure 1 illustrates the interaction of the "Bootstrapping Trust" enabler with other relevant components. Further, the two use cases introduced in Deliverable 3.2 and prototyped in this release can be observed in

the same illustration: the integrity of the virtual switch is attested by the OVS TEE, using the measurements made by the Linux *Integrity Measurement Architecture* (IMA) subsystem [5] and the expected values (currently configured in the code of the OVS TEE). The integrity of the OVS TEE is in turn attested by the attestation application residing on the controller.

The IMA is an open source trusted component included in the Linux kernel, which aims to detect file modifications (either malicious or accidental), appraise a file's measurement against an expected value, and enforce local file integrity.

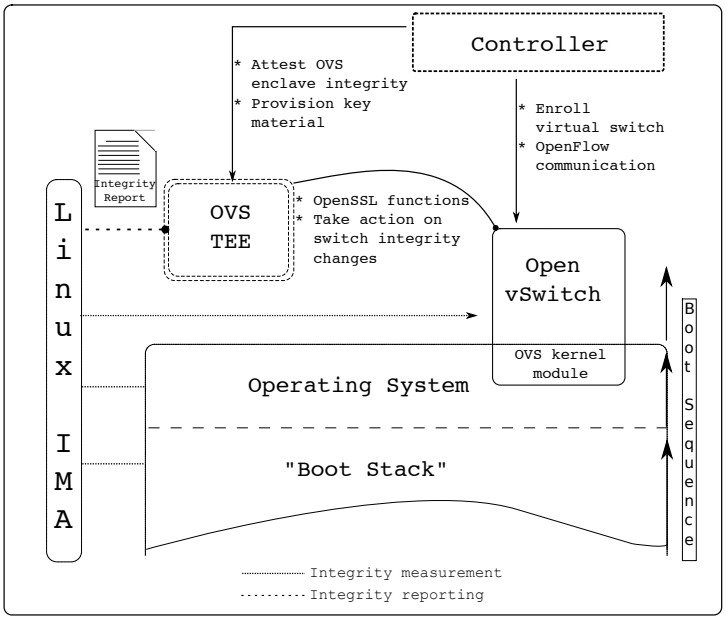

Figure 1 Interaction of the Bootstrapping Trust enabler with the other components

2.1 System Requirements

Following the system architecture presented in Figure 1, we next describe the system requirements for the current enabler. Note that the placement of the controller, relative to the rest of the aforementioned elements of the enabler (i.e. on the same host or on a different host), is implementation-specific and not relevant for the purpose of the installation. The requirements for the current enabler depend in turn on the requirements of the enabler components:

1. Ryu SDN controller, extended with an implementation of the TEE remote attestation functionality. It manages the SDN network and interacts with the Open vSwitch TEE in order to provision switch-specific credentials.
2. Modified Open vSwitch binary, running on a Linux platform with the IMA enabled. Enabling IMA is necessary in order to measure specific components of the Open vSwitch on the platform and later attest their integrity, by comparing the measurements with the expected values.
3. Emulation of Intel Software Guard Extensions (SGX) [3], an extension of the x86 Instruction Set Architecture. SGX introduces support for TEEs with remote attestation capabilities, which – once instantiated – cannot be modified by the underlying OS. The emulation is necessary to address the lack of hardware or firmware support for SGX on the platform, and is implemented using Open SGX [4]. Our current intent is to replace, in Release 2 of the enabler, the OpenSGX emulator with the recently released SGX SDK [9].

The above components require the presence of a Linux-based OS – such as Ubuntu 14.10 or above – for installation and operation. In addition to the above components, the underlying platform must provide support for the Linux Integrity Measurement Architecture [5]. According to its manual, this kernel integrity subsystem detects if files have been accidentally or maliciously altered, both remotely and locally, appraises a file's measurement against expected values stored as an extended attribute, and enforces local file integrity [10].

In addition to the standard library dependencies for the Ryu SDN controller and Open vSwitch, of the OpenSGX emulator requires the presence of the <qemu> and <libelf-dev> libraries (and underlying dependencies).

2.2 Enabler Installation and Setup

The enabler configuration consists of three main steps: OS-level configuration, emulation environment configuration, SDN configuration. We describe the installation and configuration procedures in the below sequence. We assume the host operating system is Ubuntu 14.10 and above.

2.2.1 OS-level Configuration

Integrity Measurement Architecture must be enabled and configured in order to obtain the initial measurement of the Open vSwitch binary. In Ubuntu 14.10 and above IMA is enabled in the Kernel by default. To verify that IMA is enabled, execute the following commands:

```
mount -t securityfs security/sys/kernel/security
ls /sys/kernel/security/integrity/ima
```

If the 'ima' directory exists, then the IMA is enabled.

Next, the 'ima_tcb' flag must be added to the to the boot options in order to collect IMA measurements:

```
vi /etc/default/grub
> GRUB_CMDLINE_LINUX_DEFAULT="quiet splash ima_tcb ima=on"
```

The default IMA policy will measure all executed programs (files for which the exec system call has been invoked), as well as files or devices that have been mapped to the memory for execution (using the mmap system call), and all files opened for read by uid=0. The measurements considered relevant for this enabler, on a Ubuntu 15.04, Linux Kernel 3.19.0-51-generic are:

- /lib/modules/3.19.0-51-generic/kernel/net/openvswitch/openvswitch.ko
- /etc/default/openvswitch-switch
- /lib/systemd/system/openvswitch-switch.service
- /lib/systemd/system/openvswitch-nonetwork.service
- /usr/share/openvswitch/scripts/ifupdown.sh
- /etc/default/openvswitch-switch
- /usr/share/openvswitch/scripts/ovs-ctl
- /usr/share/openvswitch/scripts/ovs-lib
- /usr/share/openvswitch/vswitch.ovsschema
- /var/lib/openvswitch/conf.db
- /usr/sbin/ovs-vswitchd

- `/etc/openvswitch/system-id.conf`

The measurements collected by IMA at boot time are found in `/sys/kernel/security/ima/ascii_runtime_measurements`
For release 1 of the enabler, only one measurement – namely that of the *Open vSwitch kernel module*[1] is attested by the TEE code. This feature is not configurable. Release 2 of the enabler will include a more comprehensive attestation.

2.2.2 Emulation environment configuration

The OpenSGX emulation environment is installed and configured according to the manual in [4]. Assuming the prerequisites are installed, the following commands will build the OpenSGX environment:

```
cd qemu
./configure-arch
make -j $(nproc)
cd ..
make -C libsgx
```

Next, from the enabler repository check out the file 'stream_ssl_enclave.c', place it into the directory '`opensgx/user/demo/`', and compile the code from the root `opensgx` directory:

```
make -C user
```

2.2.3 SDN configuration

In this section we describe the installation of the Open vSwitch binary modified to interact with the TEE emulated in Open SGX, as well as the configuration of the Ryu controller running a custom application that attests the integrity of the TEE and provisions key switch-specific key material.

2.2.3.1 Open vSwitch installation and configuration

To install and configure the modified open vSwitch binary, download the respective package from the enabler repository and run:

```
dpkg –i openvswitch-tswitch
```

All other configuration commands are identical to the Open vSwitch installation manual [5].

Next, launch the modified Open vSwitch instance on the OVS host, with the desired configuration. For the current enabler, the configuration options described in the Open vSwitch manual [7] are sufficient. To launch and configure the modified OVS instance, run the configure-ovs.sh script (included in the enabler distribution):

```
sudo sh configure-ovs.sh
```

Once the OVS instance has been configured the one must launch the custom TEE application. As mentioned above, due to the insufficient software support for SGX in Linux at the moment, the TEE application relies

[1] `/lib/modules/3.19.0-51-generic/kernel/net/openvswitch/openvswitch.ko`
[2] For simplicity, the key is supplied along with the code. A new key can be generated by running: './opensgx –k'

on an emulator. The emulator code, along with additional source code part of the enabler implementation (files "`steam-ssl.c`" and "`controllerapp.c`"), is provided in the enabler distribution.

To launch the OVS TEE, execute the following:

```
cd ~./opensgx
./opensgx -c user/demo/stream-ssl.c (this step creates the file "stream-ssl.sgx")
./opensgx -s user/demo/stream-ssl.sgx --key sign.key (this step creates the file "stream-ssl.conf")
sudo ./opensgx -i user/demo/stream-ssl.sgx user/demo/stream-ssl.conf
```

The commands above *compile* the OVS TEE application, *sign it* using the emulator-generated key[2], and *launch it* in the emulated environment.

2.2.3.2 Controller installation and configuration

To generate the PKI material for the controller and the virtual switch, run the 'gen-pki.sh' script on the controller host:

```
sudo sh gen-pki.sh
```

The enabler relies on the Ryu SDN controller framework [6]. To launch the custom controller application required by the enabler, run:

```
sudo ryu-manager --ctl-privkey ctl-privkey.pem --ctl-cert ctl-cert.pem --ca-certs cacert.pem --verbose ryu/ryu/app/attest_and_switch.py
```

Launching the 'attest_and_switch.py' controller application will also launch the "controllerapp.c" program. Following the SGX attestation protocol, this program attests the integrity of the OVS TEE application (which in turn verifies the integrity of the OVS kernel module).

2.3 Troubleshooting

Both of the SGX-related applications executing in the emulated environment (`stream-ssl.c`, `controllerapp.c`) may occasionally crash due to unexpected errors or bugs that have not been found yet. In that case, they should just be restarted, on their respective hosts:

```
sudo ./opensgx -i user/demo/stream-ssl.sgx user/demo/stream-ssl.conf
sudo ./opensgx -i user/demo/controllerapp.sgx user/demo/controllerapp.conf
```

3 User and Programmer Guide

The "Bootstrapping Trust" enabler is a meant to strengthen the security of the SDN infrastructure itself, rather than of any particular instance of the software defined network implemented in that infrastructure. It adds additional, optional integrity verification mechanisms to the infrastructure. Such mechanisms allow *detecting* changes in the integrity of the virtual switches (compared to a certain "known good value") and *preventing* integrating such switches into the *data plane* view of the controller. Obtaining the "known good value" necessary for the integrity evaluation of the switch is implementation-specific and is out of the scope of the current enabler. Note that the enabler does not provide any guarantees regarding the security

[2] For simplicity, the key is supplied along with the code. A new key can be generated by running: './opensgx –k'

properties of the attested switch implementations – this can, and should be done through the security testing of the product. Rather, the "Bootstrapping Trust" enabler merely allows verifying that the *integrity* of the switches has not been violated.

3.1 User Guide

3.1.1 Overview

In a typical usage scenario, the installed, configured and launched "Bootstrapping Trust" enabler does not require any recurrent user interaction. The current release implements only rudimentary functionality and serves as a basic proof of concept. Thus, the code of the enabler itself does not provide any configuration options (configurability is planned for release 2). However, other components on which the enabler relies – in particular, Linux IMA, offer by default extensive integrity measurement configuration capabilities. We refer the reader to [5] for a detailed description of the configuration options of Linux IMA.

The regular workflow of the enabler is presented in Figure 2. The enumerated steps shown in Figure 2, that are not in **bold** have a very limited implementation in the current release, and will be addressed in more depth in Release 2. The workflow of the enabler is as follows: at boot time (in steps **1c**), the Linux Integrity Measurement Architecture (IMA) is used to measure the components of Open vSwitch and report it to the IMA runtime measurements file[3] [4] (the specific components to be measured depend on the configuration of the IMA policy)**.** In step (**2**), the OVS TEE attests the measurement collected in step (**1c**).

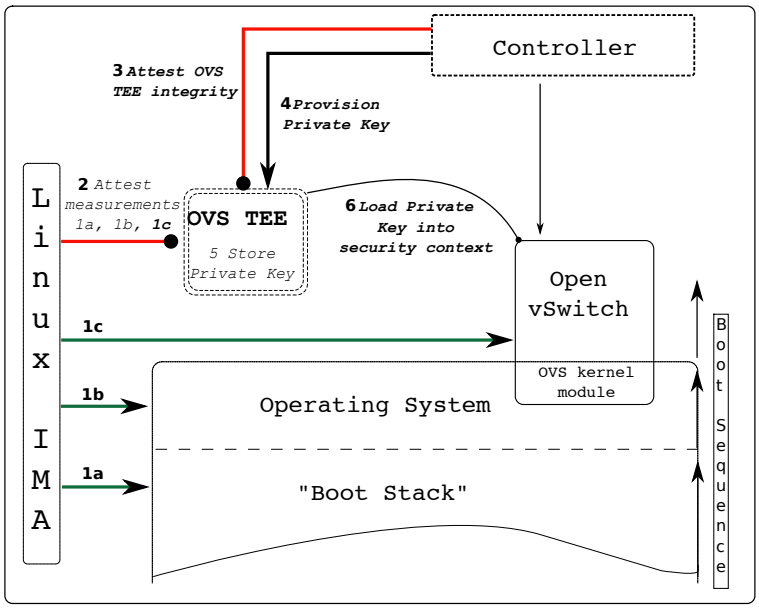

Figure 2: Workflow of enabler interaction with components on the host.

In case the local integrity attestation of the Open vSwitch component reveals a mismatch against the expected value, the OVS TEE reports the error – along with the mismatching value – to the controller. No key material is transferred to the OVS TEE; no other behavior of the controller is defined for this scenario.

Otherwise, the OVS TEE requests the *Open vSwitch private key* from the Controller; in turn, the Controller attests the integrity of the OVS TEE (**3**).

[3] We do not assume the presence of a Trusted Platform Module at this point.
[4] `/sys/kernel/security/ima/ascii_runtime_measurements`

Upon a successful attestation, the controller provisions the private key to the OVS TEE (**4**), enrolls the virtual switch in its topology (**6**) and communicates using the OpenFlow protocol; communication is protected with TLS, using the key material provisioned to the OVS TEE. Since the key will be stored within the security perimeter of the TEE, this will prevent the impersonation of the virtual switch[5] **even in the case when the adversary compromises the security of the platform where the virtual switch is executing[6]**.

In case the remote integrity attestation of the OVS TEE fails, the behavior of the controller is undefined.

3.1.2 Enabler Output

As no interaction is expected between the "Trusted Boot" enabler and the network operator, the current implementation of the enabler only outputs log messages. Such messages are produced at the following points: Open vSwitch attestation by the OVS TEE, integrity attestation of the OVS TEE by the controller; provisioning of key material to the OVS TEE; enrollment of the virtual switch instance into the topology.

3.1.3 Open vSwitch enrollment

Once the attestation and key provisioning has been done (as described in 2.2.3.1, 2.2.3.2), the virtual switch will communicate with the Ryu network controller over a protected channel, without the risk of being impersonated, even if the host operating system is compromised[7].

3.2 Programmer Guide

The enabler is not programmable.

4 Unit Tests

4.1 Information about Tests

The enabler contains several unit tests in order to test the functionality of using key material stored in the TEE. These are relevant only for *release 1* and will be superseded by a more comprehensive test harness in *release 2*. The tests have been added to the collection of tests made part of the source distribution of Open vSwitch. To run the individual tests relevant to the "Bootstrapping Trust" enabler tests, check out the source of `openvswitch-tswitch`, and following the build steps described in [7], run:

```
make check TESTSUITEFLAGS=<TestNr>
```

where <TestNr> corresponds to the test numbers specified below.

[5] This risk is mentioned in the Open vSwitch TLS configuration manual [8].

[6] As mentioned in [2] **the emulator does not provide any security guarantees** and is meant to be a proof of concept. However, such security guarantees can be provided once the enabler is implemented using the SGX SDK recently released by the CPU vendor.

[7] At this point, the TEE application does not do *runtime* monitoring of the virtual switch kernel module, so runtime attacks are not included in this scenario.

4.2 Unit Test 2252

This test aims to check that the integrity attestation of the `openvswitch.ko` kernel module in the OVS TEE, followed by the integrity attestation of the OVS TEE itself by the "Controllerapp" and the following provisioning of the private key is successful.

Test scenario:

- OVS TEE attests OVS kernel module.
- Controller app attests the integrity of the OVS TEE.
- Controller app provisions the private key to the OVS TEE.

Expected result: Success

Sample Test Output:

```
set /bin/bash './tests/testsuite' -C tests
AUTOTEST_PATH=utilities:vswitchd:ovsdb:vtep:tests::ovn/controller-
vtep:ovn/northd:ovn/utilities:ovn/controller 2252; \
"$@" || (test X'' = Xyes && "$@" --recheck)
## ------------------------------ ##
## openvswitch 2.5.90 test suite. ##
## ------------------------------ ##
2252: Test enclave attest and provision               ok

## ------------- ##
## Test results. ##
## ------------- ##

1 test was successful.
```

4.3 Unit Test 2253

This test aims to check that once provisioned, the OVS can interact with the OVS TEE to load the key into the openssl security context and establish connectivity with the controller over TLS.

Test scenario:

- Use the key obtained as a result of the integrity attestation
- Configure connectivity between Open vSwitch and controller over TLS
- Test configured connectivity between virtual switch and controller

Expected result: Success

Sample Test Output:

```
set /bin/bash './tests/testsuite' -C tests
AUTOTEST_PATH=utilities:vswitchd:ovsdb:vtep:tests::ovn/controller-
vtep:ovn/northd:ovn/utilities:ovn/controller 2253; \
"$@" || (test X'' = Xyes && "$@" --recheck)
## ------------------------------ ##
## openvswitch 2.5.90 test suite. ##
## ------------------------------ ##
2253: Test OVS-Controller communication over TLS with provisioned private key ok

## ------------- ##
## Test results. ##
## ------------- ##

1 test was successful.
```

5 Abbreviations

5G-PPP	5G Infrastructure Public Private Partnership
OVS	Open vSwitch
TEE	Trusted Execution Environment
IMA	Integrity Measurement Architecture
SGX	Software Guard Extensions
SDK	Software Development Kit
TLS	Transport Layer Security
OS	Operating System

6 References

[1] 5G ENSURE Deliverable D3.1, “5G-PPP security enablers technical roadmap (early vision)”

[2] 5G ENSURE Deliverable D3.2, “5G-PPP security enablers open specifications (v1.0))”

[3] Jain, Prerit, et al. "OpenSGX: An Open Platform for SGX Research."*Proceedings of the Network and Distributed System Security Symposium, San Diego, CA*. 2016.

[4] Open SGX project website: https://github.com/sslab-gatech/opensgx/blob/master/README.md

[5] Linux IMA Manual: https://sourceforge.net/p/linux-ima/wiki/Home/

[6] Ryu SDN controller framework: https://osrg.github.io/ryu/

[7] Open vSwitch installation manual: https://github.com/openvswitch/ovs/blob/master/INSTALL.md

[8] Open vSwitch TLS configuration manual:

https://github.com/openvswitch/ovs/blob/master/INSTALL.SSL.md

[9] Intel SGX SDK release notes, Intel Corp.

[10] Integrity Measurement Architecture Project Website: https://sourceforge.net/projects/linux-ima/