AES Lab

TPM 2.0 exploitation examples

Authors: André Zúquete and Paulo C. Bartolomeu

Version log:

- 1.0: Initial version

1 Introduction

The goal of this laboratory guide is to experiment a TPM 2.0 device. For this, we will not rely on real TPM hardware, but rather on simulation support that is helpful for development scenarios.

The exercises will be conducted on a Docker ready-to-use development environment for TPM modules provided by TPM.dev. TPM.dev is "*a place for developer-friendly computer security of IoT, Edge and Cloud systems and a group of active developers who want to make applications trusted using hardware based security*". The TPM.dev Docker image contains:

- IBM TPM 2.0 Simulator
- TCG compliant TPM2 Software Stack
- TCG compliant TPM2 Resource Manager
- TCG compliant TPM2 Tools
- wolfTPM 2.0 stack for embedded systems

2 Setup

Open Play with Docker or run the following commands on your system if you have Docker installed. Otherwise, set up Docker using the instructions at the Install Docker Engine on Ubuntu tutorial, for example.

2.1 Container and TPM server launch

```
# Download and run the tpmdev/tpm2-runtime
aes-lab:~$ docker run --name tpmlab -w ~/lab -it -d --platform=linux/amd64 -e TPM2TOOLS_TCTI=mssim:host=localhost,port=2321 \
tpmdev/tpm2-runtime:latest
```

NOTE: *If you want to run docker as non-root user then you need to add it to the docker group. First, create the docker group if it does not exist:* `sudo groupadd docker`*. Then, ddd your user to the docker group:* `sudo usermod -aG docker $USER`*. Reboot and check if docker can be run without root:* `docker run hello-world`*.*

TPM tools use an existing hardware device (`/dev/tpmrm0`) by default. To use the simulator instead, an environment variable containing `TPM2TOOLS_TCTI=mssim:host=localhost,port=2321` needs to be set, as described.

The next step is to attach the local shell to the running container. This can be done with the following command:

```
# Enter interactive shell
aes-lab:~$ docker attach tpmlab
```

The simulator then needs to be started and and the TPM 'device' initialized:

```
# Start the TPM 2.0 simulator
aes-lab:~$ tpm_server >tpm.log &
 tpm2_startup -c
```

2.2 Testing

Verify the TPM 2.0 simulator is working

```
# Obtain the next 8 octets from the random number generator
aes-lab:~$ tpm2_getrandom 8 > randombytes
```

```
# List the obtained octets
aes-lab:~$ xxd randombytes
```

Run the following command to get the groups of capabilities of the TPM device that can be listed:

```
# Query the TPM for it's capabilities / properties and print them to the console
aes-lab:~$ tpm2_getcap -l
```

A list of capability groups will be shown, as follows:

- **algorithms**: Display data about supported algorithms.
- **commands**: Display data about supported commands.
- **pcrs**: Display currently allocated PCRs.
- **properties-fixed**: Display fixed TPM properties.
- **properties-variable**: Display variable TPM properties.
- **ecc-curves**: Display data about elliptic curves.
- **handles-transient**: Display handles about transient objects.
- **handles-persistent**: Display handles about persistent objects.
- **handles-permanent**: Display handles about permanent objects.
- **handles-pcr**: Display handles about PCRs.
- **handles-nv-index**: Display handles about NV Indices.
- **handles-loaded-session**: Display handles about both loaded HMAC and policy sessions.
- **handles-saved-session**: Display handles about saved sessions.

List each of them, for instance:

```
# Check a specific capability / property
aes-lab:~$ tpm2_getcap properties-variable
```

One of these properties, `TPM2_PT_HR_TRANSIENT_AVAIL`, provides the maximum number of transient handlers that is supported (only 3). We will see the impact of this limitation in some exercises further ahead.

3 Encryption and decryption with a symmetric key

Imagine you want to maintain an encrypted content (say, a password file) that you want to be able to encrypt and decrypt without ever disclosing the key to the programs that use it. Can we do it with TPM?

The answer is, yes, we can.

First, we need to create a primary key pair in a hierarchy. We can use, for that end, the Endorsement hierarchy, which is targeted to the computer user:

```
#  Create a primary object under one of the hierarchies: Owner, Platform, Endorsement, NULL.
aes-lab:~$ tpm2_createprimary -C e -p top -G rsa2048 -c primary.ctx
```

This command creates in the TPM a transient, primary key pair. Transient means that it disappears upon a TPM reset. The meaning of the parameters is the following:

- `-C e`: use the (e)ndorsement hierarchy. Other alternatives could be the (o)wner, the (p)latform or the n(ULL) hierarchies;
- `-p top`: the authentication credential (the `top` password) that must be provided in an session to use this primary key;
- `-G rsa2048`: the type of algorithm this key is targeted to (RSA) and the size of key (`2048`-bit modulus);
- `-c primary.ctx`: a context file that allows to recreate this key in the TPM without providing all these parameters.

Upon executing this command, the newly created key is kept transiently kept inside the TPM, and accessible with the handle `0x80000000`. You can see this with the command:

```
# Check a specific capability / property
aes-lab:~$ tpm2_getcap handles-transient
```

Note that the key can be flushed from the TPM because it can be recreated with the file produced during its creation:

```
# Remove a specified handle, or all contexts associated with a transient object, loaded session or saved session from the TPM.
aes-lab:~$ tpm2_flushcontext 0x80000000
```

```
# Check a specific capability / property
aes-lab:~$ tpm2_getcap handles-transient
```

Let us know create a symmetric key protected under the primary, asymmetric key just created. Primary keys are always storage keys, which means they only serve to protect its child keys (of which it is a parent).

```
# Create a child object. The object can either be a key or a sealing object.
aes-lab:~$ tpm2_create -C primary.ctx -P top -G aes128cfb -u aes-pub.bin -r aes-priv.bin -c aes.ctx -p sec
```

This command creates a `128`-bit AES key, to be used in CFB mode. The information about the parent key is provided in the file `primary.ctx`, together with the password to be able to use it (top). The symmetric key is randomly generated by the TPM, and stored, wrapped by the primary key, in the file `aes-priv.bin`. The file `aes-pub.bin` will contain public information about this key that can be used to recreate it in the TPM. Finally, the usage of this key is protected by the password `sec`.

Upon this command, the TPM contains two transient keys: a primary, storage RSA key pair, and a symmetric, encryption/decryption AES key. By default, the AES key cannot be duplicated to another hierarchy, or to be protected under a different parent storage key, as it is created with the `PMA_OBJECT_FIXEDTPM` and `TPMA_OBJECT_FIXEDPARENT` attributes.

The creation of the AES key leaves two transient keys inside the TPM, which handles you can see with this command:

```
# Check a specific capability / property
aes-lab:~$ tpm2_getcap handles-transient
```

As the TPM resources are scarce, and it only supports 3 transient handles, we will flush them all from the TPM upon the key creation:

```
# Remove all transient objects
aes-lab:~$ tpm2_flushcontext -t
```

Now, each time we need to use the AES key, we load it first in TPM, and we can use the handle to it to require the TPM to encrypt or decrypt data with it.

```
#  Load both the private and public portions of an object into the TPM
aes-lab:~$ tpm2_load -C primary.ctx -P top -r aes-priv.bin -u aes-pub.bin -c aes.ctx
```

We can see that this command, in fact, installs two keys in the TPM: the intended AES key, and the primary storage key that is required to install it.

```
# Check a specific capability / property
aes-lab:~$ tpm2_getcap handles-transient
```

We can flush the first, as it will not be necessary:

```
# Remove a specified handle, or all contexts associated with a transient object, loaded session or saved session from the TPM.
aes-lab:~$ tpm2_flushcontext 0x80000000
```

Now, we can finally use the AES key for data encryption and decryption. On each request, we need to provide the password (`sec`) that was specified for its protection upon creation. Note that this password is validated by the TPM in order to use the key inside for encryption/decryption, and not to reveal the key in cleartext outside the TPM.

Create a small file, with an arbitrary length, named `in`. Then, encrypt it, producing an encrypted output on file `out`:

```
# Performs symmetric encryption or decryption with a specified symmetric key on the contents of File.
aes-lab:~$ tpm2_encryptdecrypt -c 0x80000001 -p sec -o out in
```

The reverse operation, for recovering the original contents of file `in` in `in.1`, is the following:

```
# Performs symmetric encryption or decryption with a specified symmetric key on the contents of File.
aes-lab:~$ tpm2_encryptdecrypt -d -c 0x80000001 -p sec -o in.1 -d out
```

You can confirm that files `in` and `in.1` are absolutely equal, using the `diff` or `cmp` commands:

```
# Compare files
aes-lab:~$ cmp in in.1
```

NOTE: *No output means equality.*

3.1 Homework

Try to create an asymmetric key pair as child of the same primary storage key, and use it to encrypt/decrypt small contents.

4 Clean up

The container can be stopped by issuing the following command:

```
#  Stop one or more running containers
aes-lab:~$ docker stop tpmlab
```

Afterward, it can be removed with this command:

```
#  Remove one or more containers
aes-lab:~$ docker rm tpmlab
```

The image used to create the container can also be removed. First, it is necessary to find the corresponding `<IMAGE ID>` using the `docker images -a` command, which shows all installed images. After, the intended `<IMAGE ID>` image can be removed by issuing the following command:

```
#  Remove one or more images
aes-lab:~$ docker rmi <IMAGE ID>
```

Bibliography

TPM 2.0 Library

Trusted Platform Module 2.0 tools documentation

TPM.dev Docker image

Install Docker Engine on Ubuntu