DVSE Walkthrough

# Introduction and the purpose of the document

This document describes a walkthrough of DVSE exploitation.

DVSE (or Damn Vulnerable SGX Enclave) is a training target for hacking SGX enclaves.

It was presented first time @Recon2017 and resides at [this github repository](https://github.com/wireshrink/RECONMTL-2017).

The presentation which resides in [this folder](https://github.com/wireshrink/RECONMTL-2017/docs) of this repository provides background about it and some additional links for the concerned fellow citizens. Reading the presentation first will ease the reading of this document significantly.

# Application and its workflow

The application consists of client, server and the Evil DRM ☺. In the current configuration variant, it provides a possibility to play some public domain cartoons. The configuration is stored in so-called EPG, or “electronic program guide”, and described below.

### Initial EPG (Electronic Program Guide) state

Here is the content of the EPG.

Popeye - i don't care,1,2,payment required,

Popeye - patriotic popeye,2,2,payment required,

Popeye - private eye popeye,3,2,payment required,

Popeye - shuteye popeye,4,2,payment required,

The big bad wolf,6,2,payment required,

woody woodpecker pantry panic,7,2,free view,

It contains 6 public domain cartoons, each cartoon is described in a single line, parameters separated by comma, where the first string is a cartoon name, movie id is a number which follows the name, and the last string defines the payment requirements.

### The movie play workflow

1. User defines connection parameters and the library directory if needed (on first run only)
2. Application initializes the SGX enclave
3. On initialization, the enclave connects to the server with usage of enclaved OpenSSL, checks the certificate of the server by its certificate’s SHA1 signature, and downloads from there 3 files
   1. Blob – template for storing local user payment data (if previous version does not exist already)
   2. Coupons – a list of coupons which allow to use so-called payed movies
   3. Epg – a list of currently available movies in the text format
4. Then user opens the EPG window, and double-clicks the movie.
5. The application asks the enclave to prepare the movie for play.
6. The enclave downloads the movie from the server and stores it encrypted in the library folder
7. The enclave is checking if one of the following conditions is met:
   1. The movie is free to play
   2. The user has enough coins to allow playing the movie now
   3. The user already paid for this movie and the grace period (2 hours specifically) did not pass yet (the current time is obtained from the outside via OCALL)
8. If one of these conditions is met, the enclave opens the corresponding media file via OCALL and became ready for playing
9. The application asks for movie chunks as it needs and plays the movie
   1. Enclave reads encrypted data, decrypts it and gives to the application

### Coupons and paying “protected movies” workflow

If the user wants to play a movie which is marked as “payment required” in the EPG {s}he needs a means to pay for it. These means are called “coupons” and described in “coupons” file on the server side. Each coupon has 32 byte length and starts from string DVSE\_CPN.

If user has a coupon in his possession {s}he can apply it via setup dialog.

When user pastes the coupon into the corresponding edit box and presses “apply” button the following happens:

1. The enclave-driving application calls an enclave in order to verify the coupon.
2. Inside an enclave the coupon is compared with the content of coupons file.
3. If one of the coupon matches the enclave checks if was used previously by comparing it with the list of previously used coupons which is stored in the blob file.
4. If it was not used previously user’s coupon balance is increased by 25 coins and the used coupon is wrote into the blob as used.
5. Coupons acquired in this way can be used for purchasing protected movies as described in previous chapter.

# Vulnerabilities inserted deliberately

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerability | How to find and exploit | How to avoid | Category |
| No replay protection on the blob file | Can be found by design review/reverse engineering. Can be exploited by simple deleting of the blob file (which will cause to creating empty one without marks of used coupons), or storing the blob file outside after applying all coupons and returning it back when needed.  See “” function in EnclaveTest project for exploitation example. | Store balance data on the server. Insert monotonic counter into the blob which will increase with each rewrite, and delete the library when somebody tampers it.  Don’t use coupons as an idea or connect coupon to the specific movie. | Bad design |
| Strcmp is used to compare coupon values | Can be found by code review/reverse engineering. Can be exploited as any other timing attack on strcmp. **Important note**: Implementations of strcmp on Windows and Linux are very much different: Windows implementation is heavily optimized and based on SSE2 instruction set extension, which makes this attack on windows much less practical. | Don’t use strcmp on sensitive data, it is evil and ill-reputed practice. | Bad code |
| Files are downloaded regardless playing allowance | Can be found by code review/reverse engineering. Cannot be exploited directly, but can (and will) be used with other bugs. See “” function in EnclaveTest project for exploitation example. | Don’t store something that you don’t know you really need. | Bad code/Bad design |
| The enclave trusts the time got via OCALL | Can be found by code review/design review/reverse engineering. Can be exploited by hijacking the timestamp before it is sent back to the enclave during corresponding OCALL execution. See “” function in EnclaveTest project for exploitation example. | Don’t trust OCALLs. In this specific case use time from server provided with via network. | Bad code/bad design |
| The enclave trusts to the file handle got via OCALL | Can be found by code review/design review/reverse engineering. Can be exploited by opening another file during OCALL execution (for example in order to make an enclave think that it opens file allowed to play). See “” function in EnclaveTest project for exploitation example. | Don’t trust OCALLs. In this specific case movie id can be embedded into the data guarded by AES GCM, for example into so called AAD, and checked on reading and opening. | Bad code/bad design |
| Decrypted coupons are stored in enclave memory decrypted all the enclave lifetime | Can be found by code review/design review/reverse engineering. Cannot be exploited solely, but if some data is leaked your secrets may be inside. See “” function in EnclaveTest project for exploitation example. | Don’t leave the secrets unattended, decrypt secrets only when they are really needed and encrypt them back afterwards. | Bad code/bad design |
| Missing index check in reading the EPG pages | Can be found by code review /reverse engineering/ fuzzing.  Can be exploited by reading the non-existing EPG pages, all the heap will probably be leaked. See “” function in EnclaveTest project for exploitation example. | Write good code, validate user input, conduct code review, fire bad programmers on time. | Bad code |
| Sealed log file write function is left by mistake in the enclave | Can be found by code review/ reverse engineering. Can be exploited by writing enough logs in order to reveal key generation material and AES GCM intermediate results that can be used for decrypting the content without knowledge about exact key value. See “” function in EnclaveTest project for exploitation example. | Write good code, validate user input, conduct code review, fire bad programmers on time.  #ifdef \_DE**N**UG is a more common mistake that it is usually considered. | Bad code |
| Type cast mistake in sealing replacement function | Can be found by code review/ reverse engineering. Can be exploited altogether with previous vulnerability: this cast mistake leads to significant key derivation material exhaustion. See “” function in EnclaveTest project for exploitation example. Note: this vulnerability was implemented with mistake that significantly hardens its exploitation – yes, 6 bytes of random data for key derivation is significantly less then 44 bytes, but it still requires a lot of time and disk space to generate intermediate GCM results with the previous vulnerability. | Write good code, validate user input, conduct code review, fire bad programmers on time. | Bad code |

# Exploitation examples walkthrough

# Mistakes and other problems made not deliberately