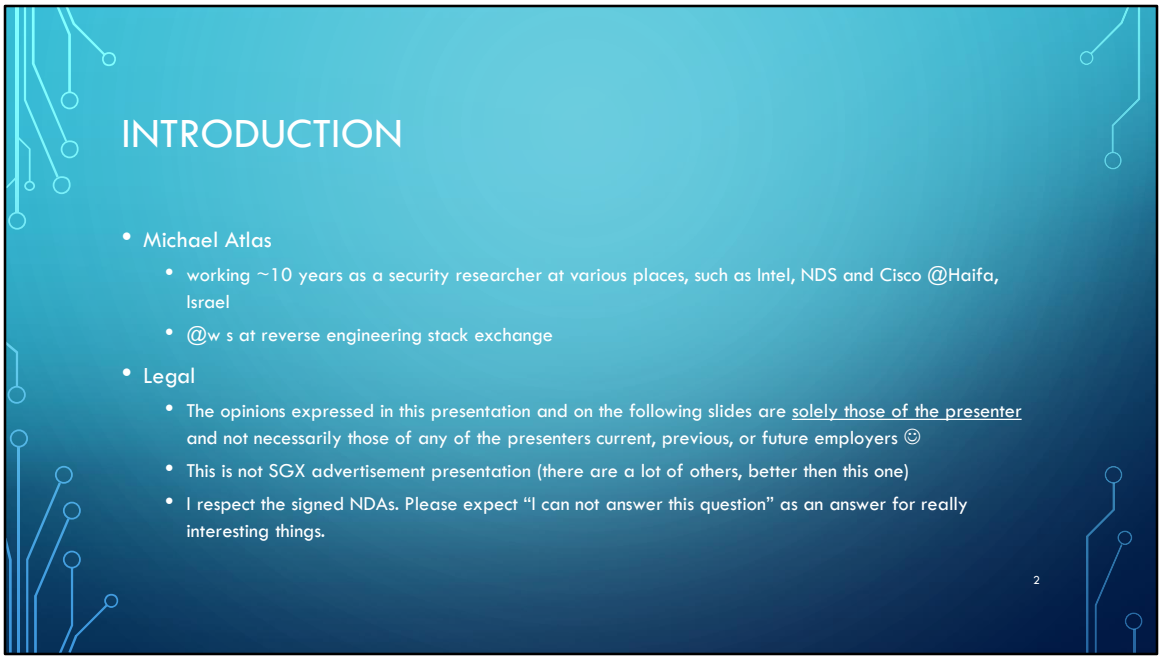
Each organization has its own dialect of English. That's why there are 2 probably not too much understandable abbreviations on this slide: SGX and WKM. SGX is a software guard extensions of Intel and by extension I mean instruction set extension. WKM is an opposite to the BKM (or best known methods) and means, of course , worst known methods.
Today I going to talk about mistakes that can be done (and usually done) with SGX enclave programming – and going to present a training target for hacking SGX enclaved software.

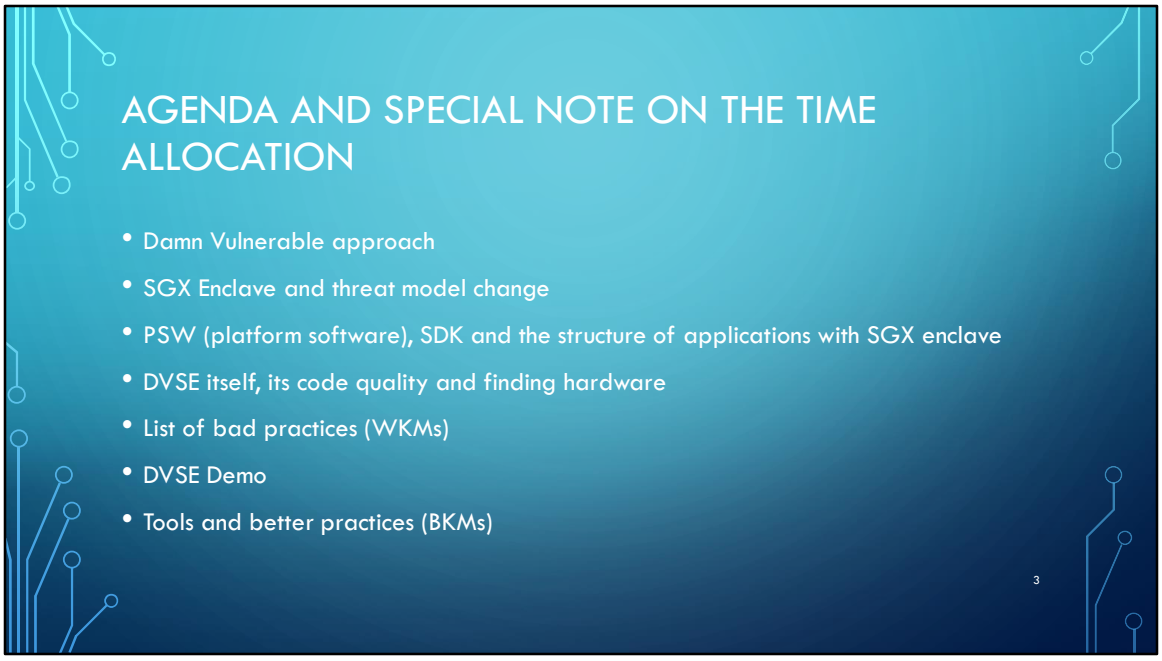


Let me introduce myself,
Michael Atlas

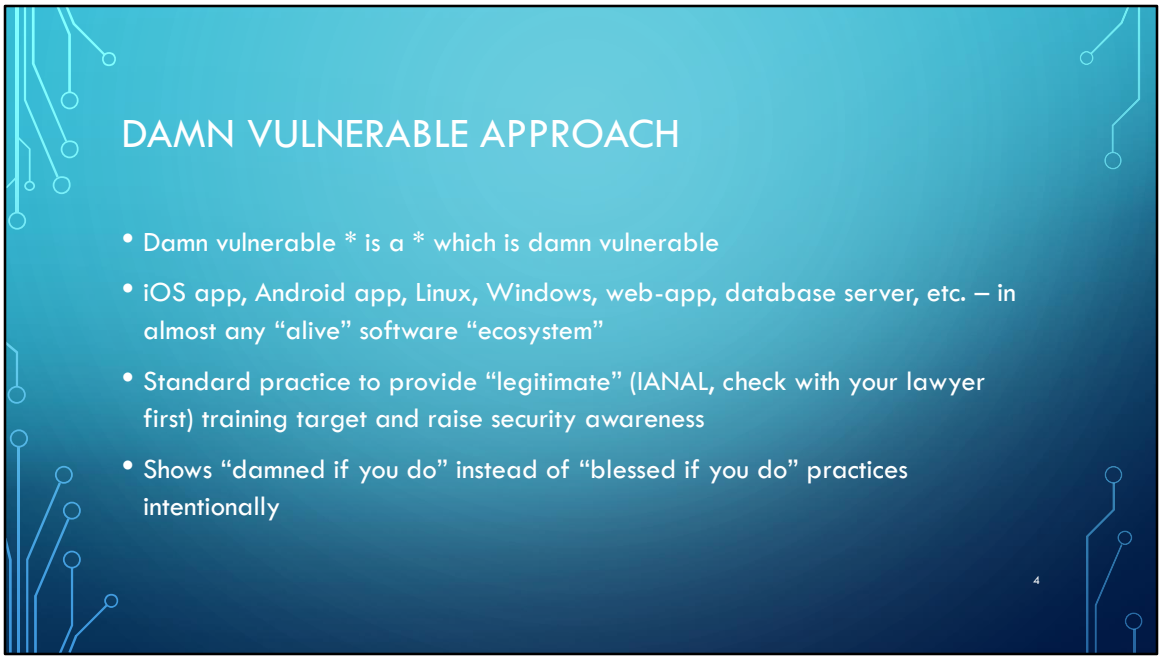
working ~10 years as a security researcher at various places, such as Intel, NDS and Cisco @Haifa, Israel
@w s at reverse engineering stack exchange

And I have to state some things loud and clear before we starting
Legal

The opinions expressed in this presentation and on the following slides are solely those of the presenter and not necessarily those of any of the presenters current, previous, or future employers ☺
This is not SGX advertisement presentation (there are a lot of others, better then this one)
I respect the signed NDAs. Please expect "I can not answer this question" as an answer for really interesting things.

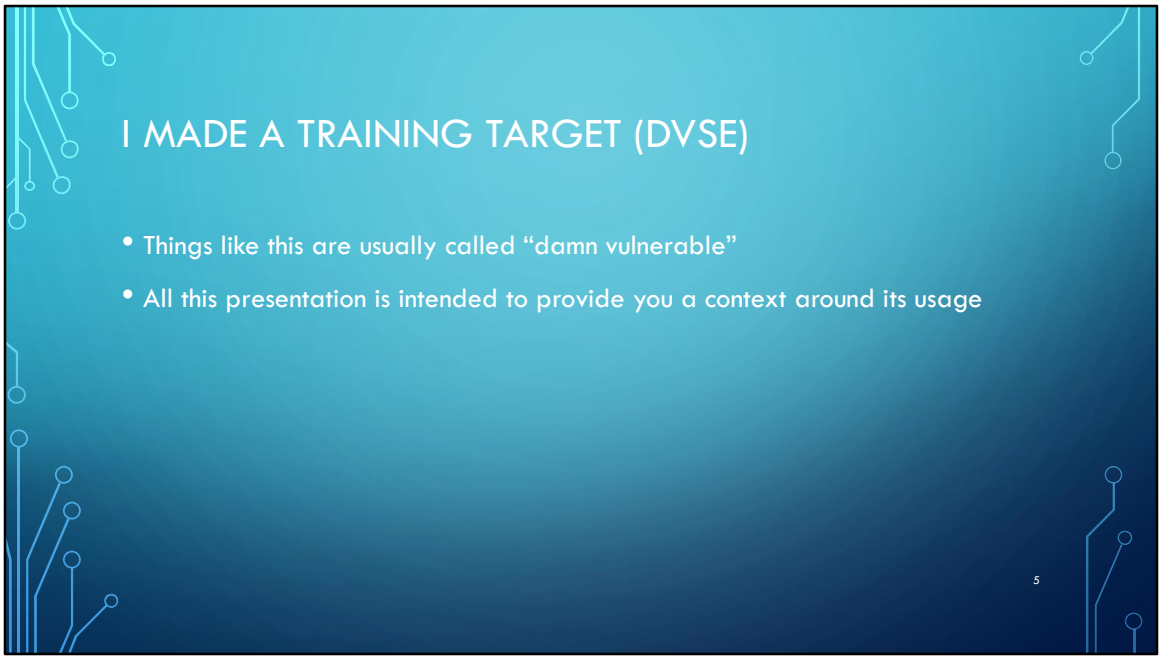


Note that SGX is a bit complicated and some of this talk will be dedicated to describing SGX itself and its SDK. There are a lot of good people that made a lot of presentations about it.
In addition I'll try to avoid spoilers in the details of specific WKMs implementation.
Here is an agenda: I going to speak about the following
-remind Damn Vulnerable approach
- What is SGX Enclave, how it changes standard threat model and why I think that it is good
-What is PSW (platform software), SDK and what is a standard structure the structure of applications with SGX enclave built with this SDK
-DVSE itself, its code quality and finding hardware
-List of bad practices (WKMs)
-DVSE Demo
-Tools and better practices (BKMs)

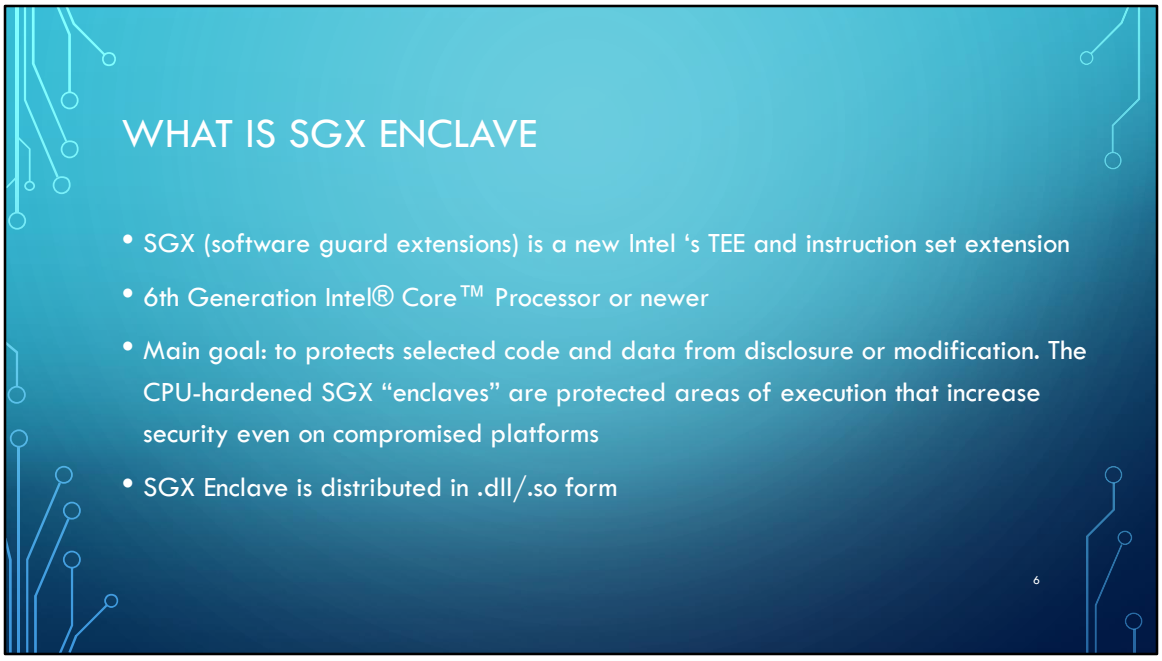


Some of you probably remember what is “damn vulnerable” exercises

-Damn vulnerable * is a * which is damn vulnerable
-iOS app, Android app, Linux, Windows, web-app, database server, etc. – in almost any “alive” software “ecosystem”
-Standard practice to provide “legitimate” (IANAL, check with your lawyer first) training target and raise security awareness
-Shows “damned if you do” instead of “blessed if you do” practices intentionally

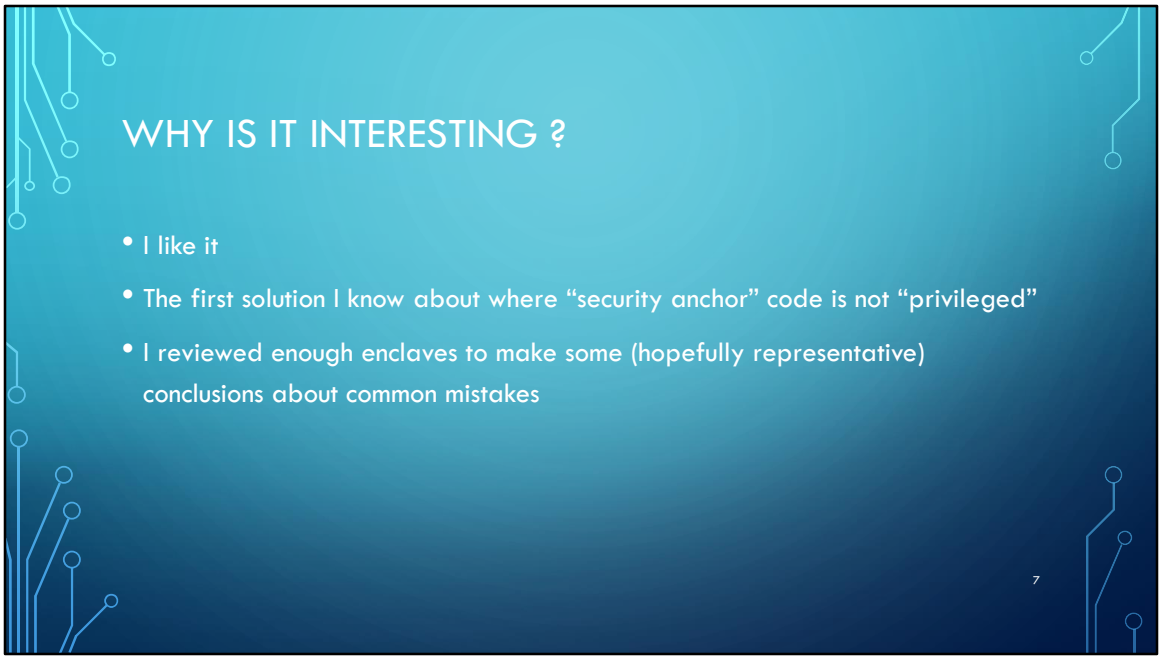


I made a thing like this for SGX enclaves
All this presentation is intended also to provide you a context around its usage
And of course I call this training target a damn vulnerable SGX enclave according to this old respectable tradition



- SGX is new Intel 's TEE and instruction set extension (2 opcodes enclu and encls, with so-called leaves numbers of which are passed in RAX register),
- Exists in 6th Generation Intel® Core™ Processor or newer (SkyLake and CabyLake for now)
- Main goal: "to protects selected code and data from disclosure or modification. The CPU-hardened SGX "enclaves" are protected areas of execution that increase security even on compromised platforms."
- SGX enclave is a blob of code, packed as .so/.dll with well defined interfaces which can be called from the usual application as functions

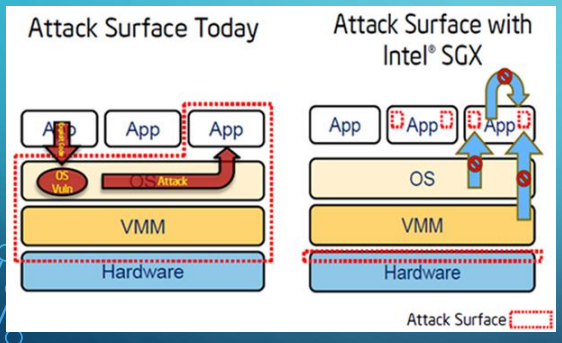
In short: SGX enclave is an isolated, signed, and attestable piece of code with well defined interfaces that allows to the programmer to execute it safely without of intervention of other parties, even most privileged - such as BIOS, OS, hypervisor, SMM and Management Engine.



First of all … I like it.
This is the first solution I know about where “security anchor” code is not “privileged”, which is definitely right thing

And I think I reviewed enough SGX enclaves during my work @Intel to be able to make some (hopefully representative) conclusions about common mistakes

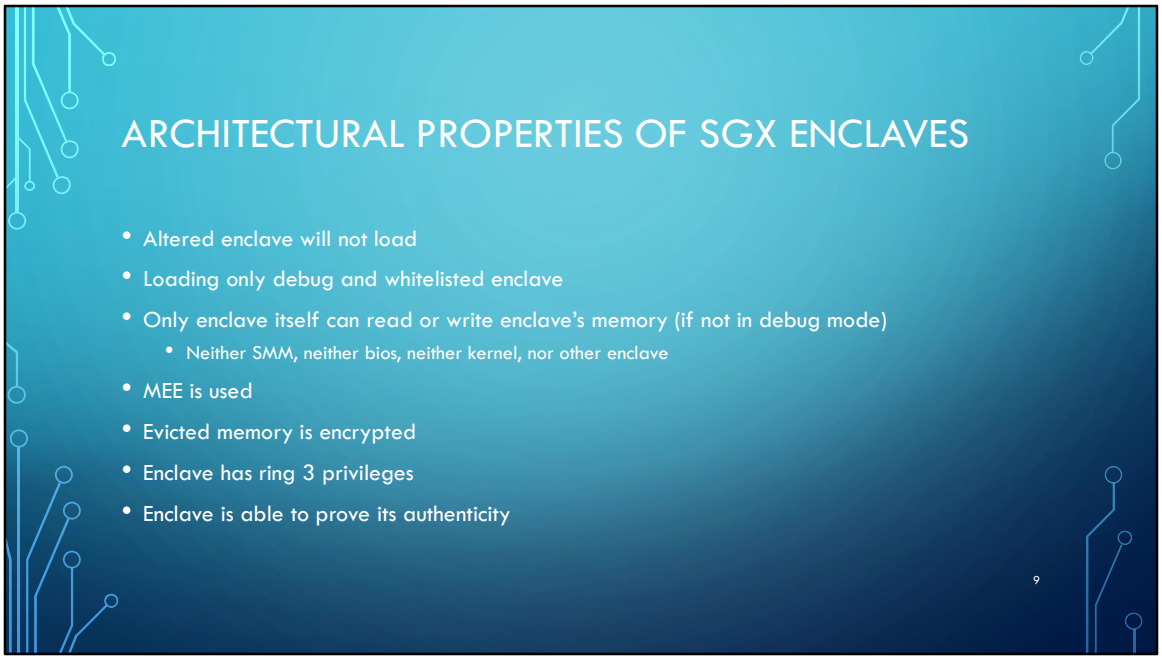
SGX: ATTACK SURFACE REDUCTION



- The attack surface of the application is reduced to defined enclave interfaces
- Privileged code can not access enclave's internal state
- Enclave is signed and attestable
- Enclave has access to unique platform+enclave specific crypto keys (and nobody else)

8

One of the main ideas behind this invention is to reduce the attack surface of the application, which means creating some kind of code and data enclave which can not be accessed by privileged software with some additional security properties. If the software is designed properly - the attack surface of the application's security anchor is reduced to defined enclave interfaces(instead of all the privileged software and all the application itself).
Privileged code can not access enclave's internal state – and other enclaves too.
Enclave is signed and attestable
In addition SGX enclave is able to use unique cryptographic keys derived from processor specific fuses, enclave measurements and some other things (which means that the same enclave will not be able to decrypt the data encrypted on other system).



These enclaves have the following architectural properties:

Altered enclave will not load (It is signed, loading fails, cryptographically protected by hardware)
Loading only debug and whitelisted enclave (licensing enclave, the part of mandatory platform woftware, will decline others)
Nobody except enclave itself can read or write enclave's memory if the enclave is not defined as debug (debugging instructions will work otherwise)
Neither SMM, neither bios, neither kernel, nor other enclave
Enclave's memory is encrypted (MEE also known as Memory Encryption Engine is used in order to avoid HW attacks)
Evicted memory is also encrypted and protected from replay attack
There is a usual design pattern – more secure code should be more privileged (and all negative numbered rings are the proof for that)
SGX designers made completely different decision - Enclave itself has ring 3 privileges, so it allegedly can not harm anyone else
Enclave is able to prove its authenticity to authorized parties

These properties define some consequences:

SGX enclave is specially designed for keeping secrets safe

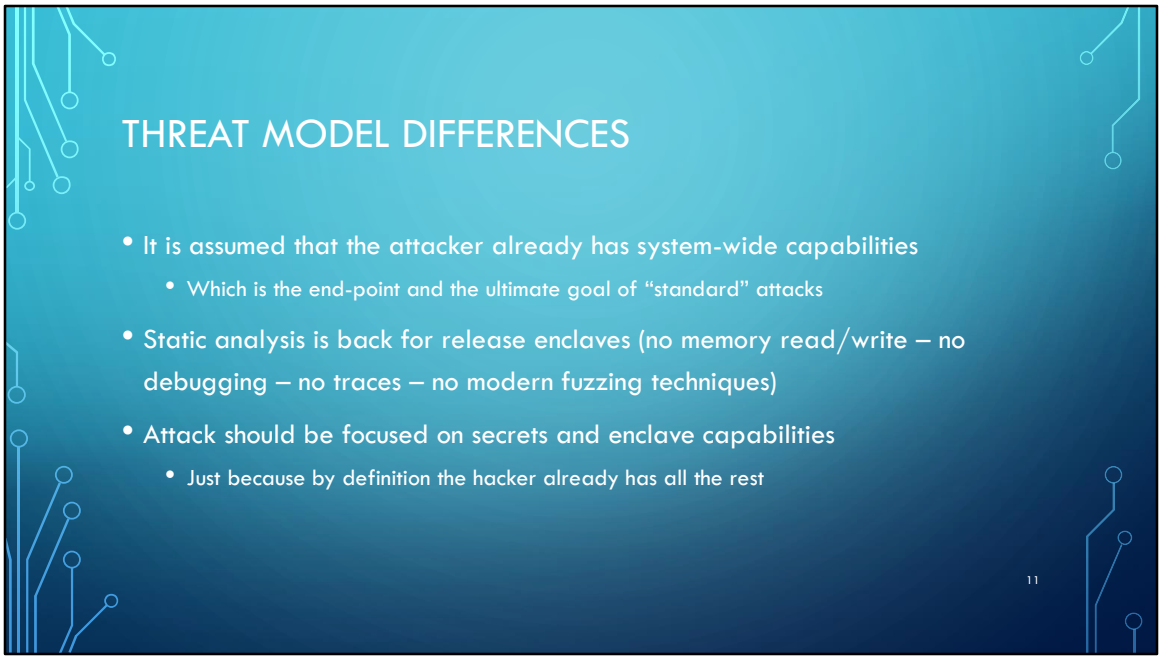
SGX enclave is a perfect place to hide and isolate interesting activity – and you can define word “interesting” as you like

SGX enclave can be very small

Smaller SGX enclave has significantly smaller attack surface

SGX enclave is a part of an ordinary application

SGX enclave is a code, written by programmers. They are making mistakes. I know that my English is bad, but I really mean that they are writing code now and making mistakes just now, in this very moment (at least in countries were Sunday is a working day) – and it is never ending process and it will continue until something will not disappear: SGX or programmers ☺ .



Having such a thing in the system makes some shifts in “standard” threat model. SGX enclaves as a feature were designed to allow correct and secure enclave functioning in worst conditions, even if the system is completely compromised by the attacker.

It is assumed that the attacker already has system-wide capabilities

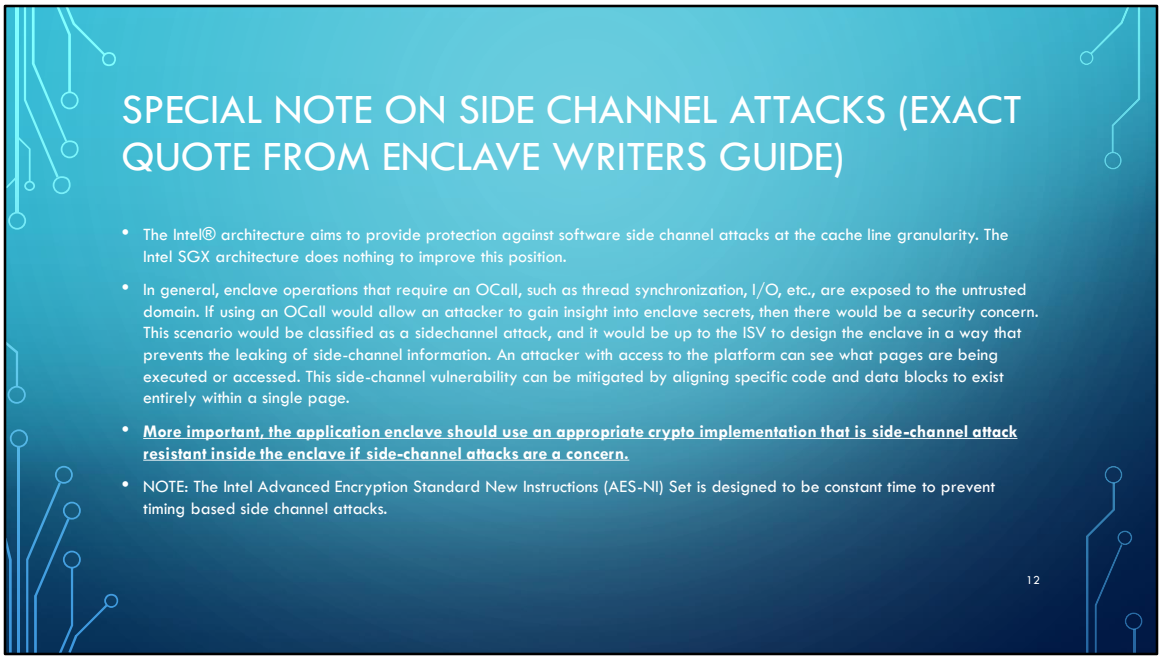
Which is the end-point and the ultimate goal of “standard” attacks

And it still holds the ground even in this case: attacks on enclaves are starting after the most ultimate standard goal of the attacker is achieved

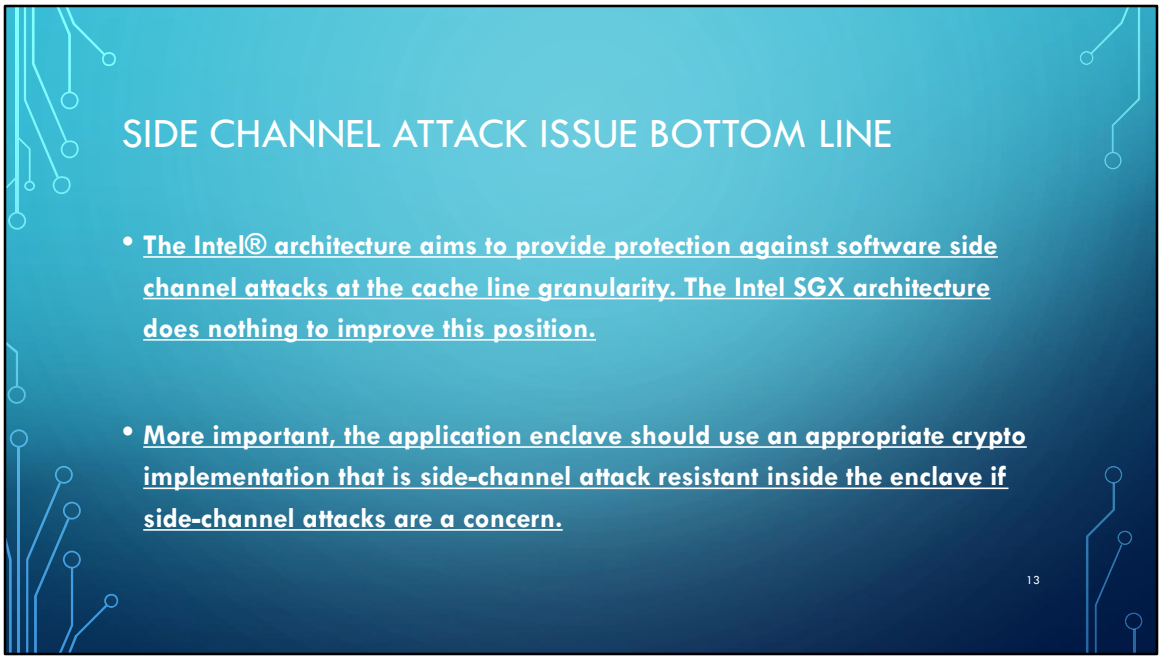
No memory access means actually no debugging. No debugging in offensive context means no smart fuzzing, which means that at least for SGX release enclaves pure static analysis is back.

It also changes the focus of the attack: really successful attacks on SGX enclaves will be mostly focused on extracting of enclaves secrets and reusing enclaves capabilities rather then code execution -

Just because by definition the hacker already has all the rest



As a part of changes in threat model – timing attacks are getting more and more important and this is a very special case – but you are more then welcome to explore this direction, it works.
There are some articles published recently about it . I keeping this long and exact quote from the SGX enclave writers guide here for those that can not hear me and will be reading the presentation. Please don't rush into reading it, the essence follows:

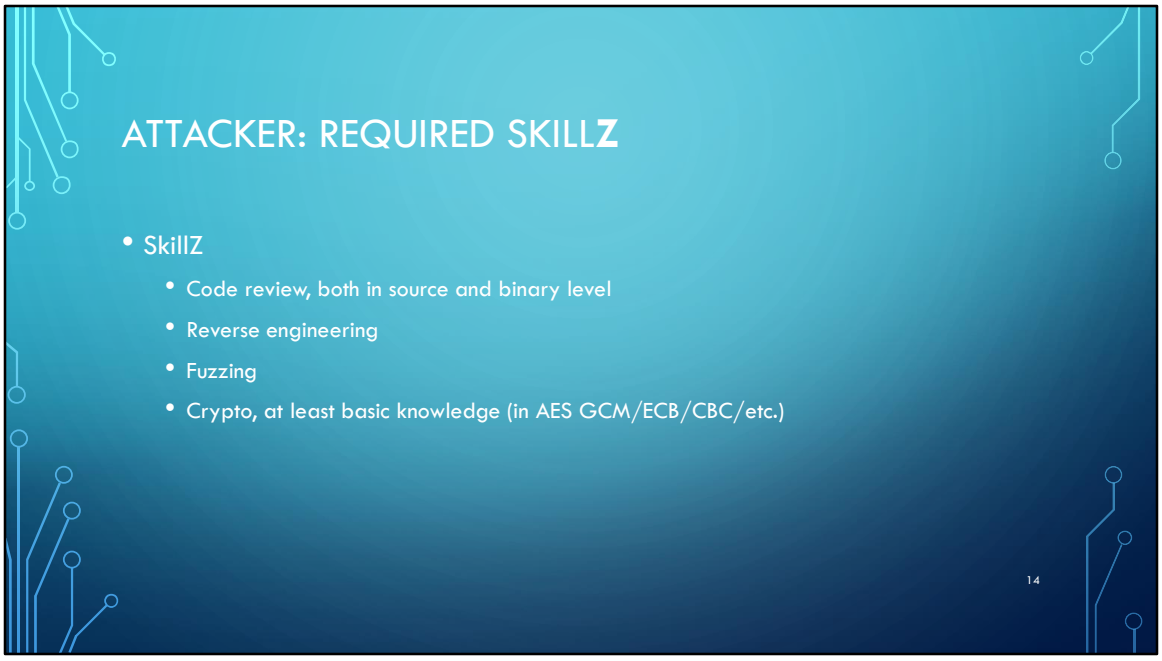


Here it is –

The Intel® architecture aims to provide protection against software side channel attacks at the cache line granularity. The Intel SGX architecture does nothing to improve this position.

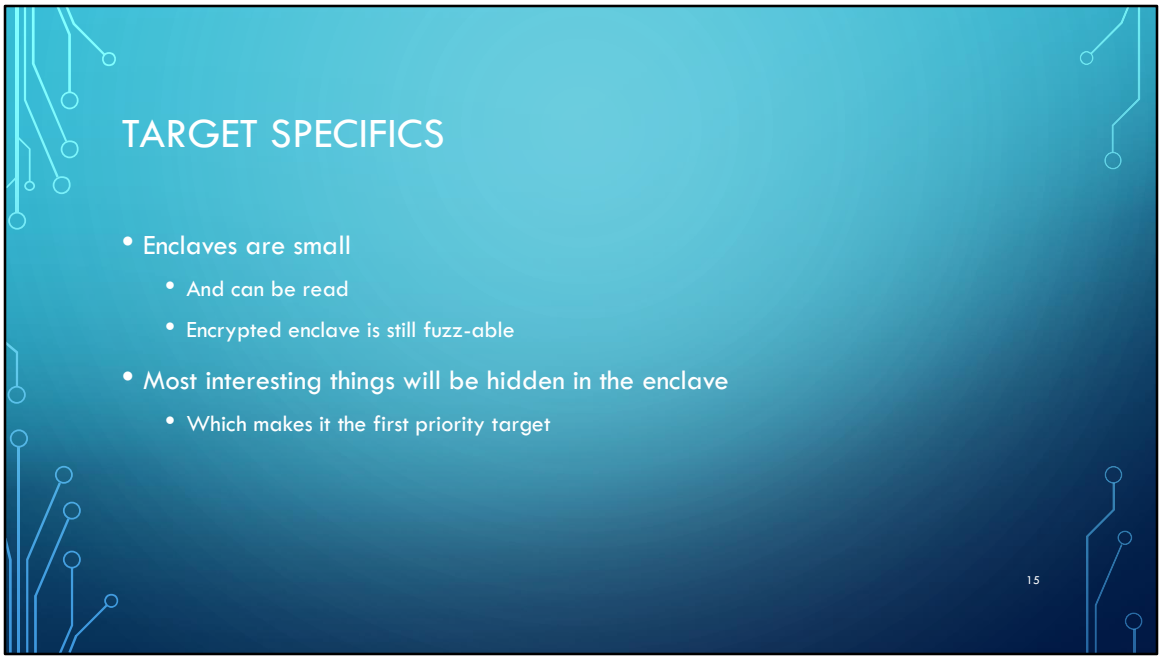
More important, the application enclave should use an appropriate crypto implementation that is side-channel attack resistant inside the enclave if side-channel attacks are a concern.

Let me translate it from English to Understandable: it is on responsibility of the programmer to care about it.



New task requires new/good old set of tools: this brings us to a bit different skillset – running fuzzer and !exploitable in WinDbg is definitely not enough.
The following skillz are required for SGX enclaves reviewing:

Code review, both in source and binary level (meaning – old good reverse engineering), will be much more productive.
Dumb fuzzing, and
Attacker will also need at least some basic knowledge in cryptography because AES GCM is used for default sealing.



The following things really make all thi situation different:

If the enclave is good – it is really small, smaller enclave means smaller attack surface

On the other side, the fact of enclave existence, reveals the most interesting target and (none accessible) location of the secrets immediately, all the “interesting” things are located at the same place, the enclave itself.

And you have no debugging capabilities inside for release enclaves.

In a standard case it is readable, and you literally can read it all.

However, enclave itself can be encrypted.

But even in this case the enclave driving application is reverse-engineer-able and blind fuzzing is still possible.

So, again, static analysis is back. Seriously.

So most interesting things will be hidden in the enclave

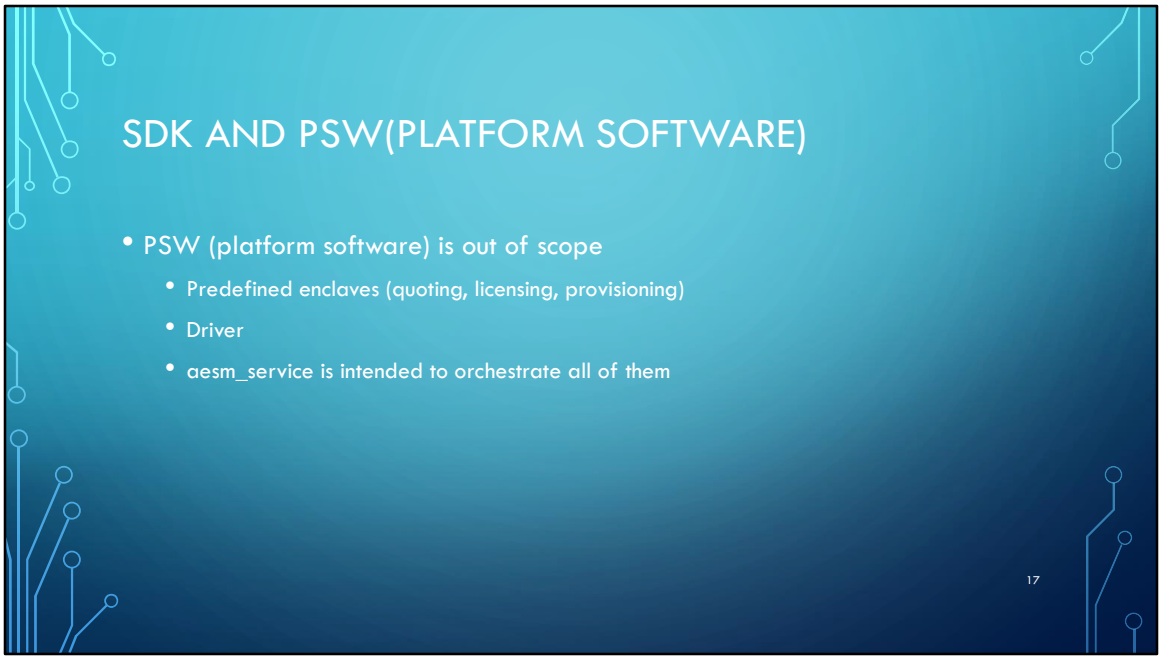
Which makes the enclave the first priority target

Now we know what enclave is, and what properties it has.
Let's explore what is required for playing with it.
First of all it appears that having processor with SGX support is not enough … (btw, by this link you'll find the full list)

http://ark.intel.com/Search/FeatureFilter?productType=processors&SoftwareGuardExtensions=true

Lars Richter (@ayeks) maintains a list of SGX enabled hardware, the link is here. This link is very useful because
"SGX is turned off by default and must enabled via MSR.IA32_Feature_Control.SGX_Enable. Only the BIOS can make changes to the IA32_Feature_Control."
For example Intel SGX SDK requires as the following for the hardware:
"Required Hardware: 6th generation Intel® Core™ processor (or later) based platform with Intel SGX-enabled BIOS support"
I spent a week for finding a laptop (this one, Dell Inspiron 15 5578 2-in-1) that really supports it.
Just try to imagine – you calling a support of specific hardware vendor and asking a

question – does this specific system supports SGX on BIOS level.
They obviously don’t understand the question, and trying to clarify it: and they don’t ask what is SGX, they ask what is BIOS ☺



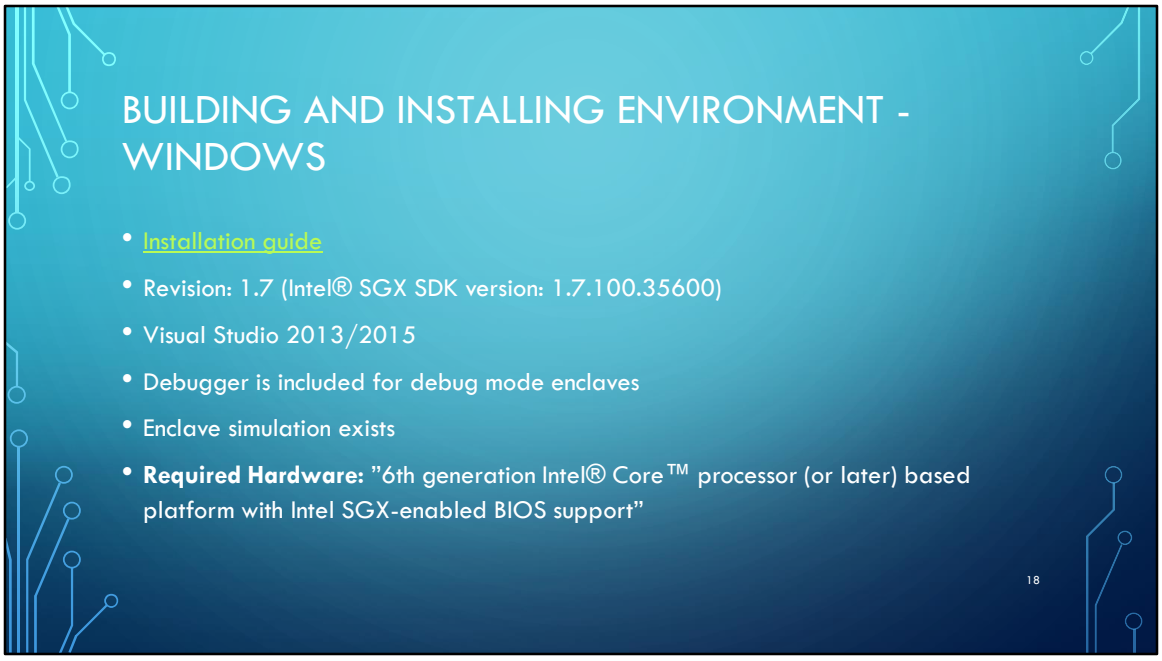
IN addition to hardware and BIOS we need some software.
Playing with SGX requires SDK and PSW which is actually out of scope of this talk …
PSW (platform software) – it is kind of runtime environment which contains driver, architectural enclaves and the aesm_service

Architectural enclaves (quoting, licensing and provisioning) are mostly related to enclave licensing and attestation features.

SDK is actually for building the enclave and the application

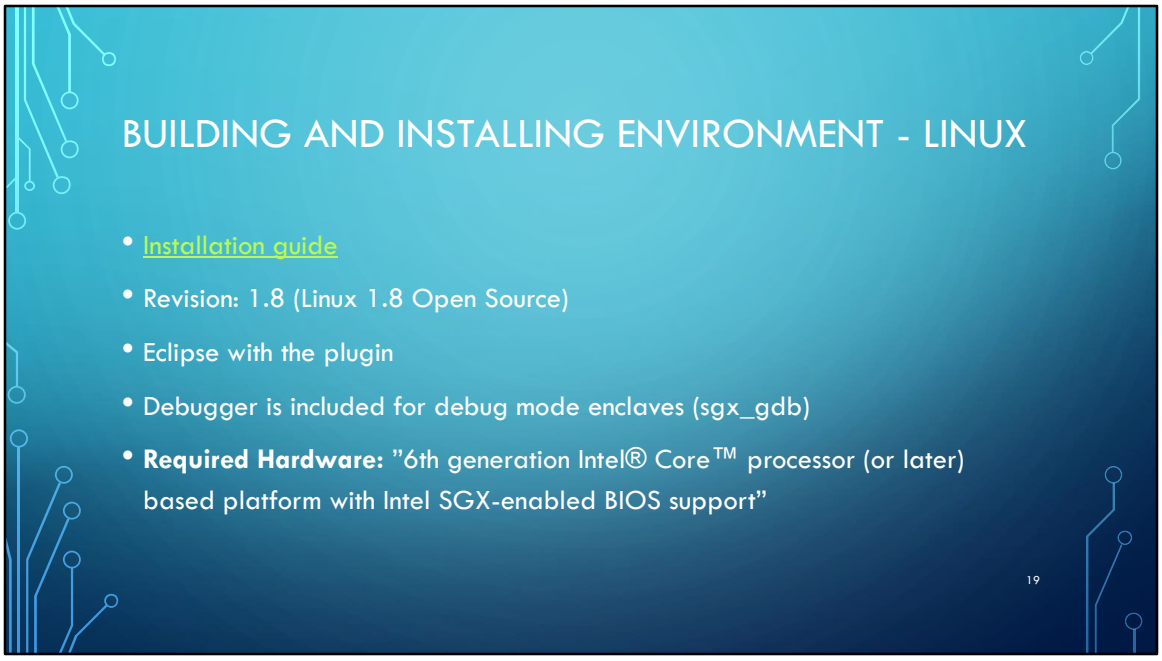
aesm_service is intended to orchestrate all of them

*** PSW is kind of runtime environment,
*** SDK is for compiling



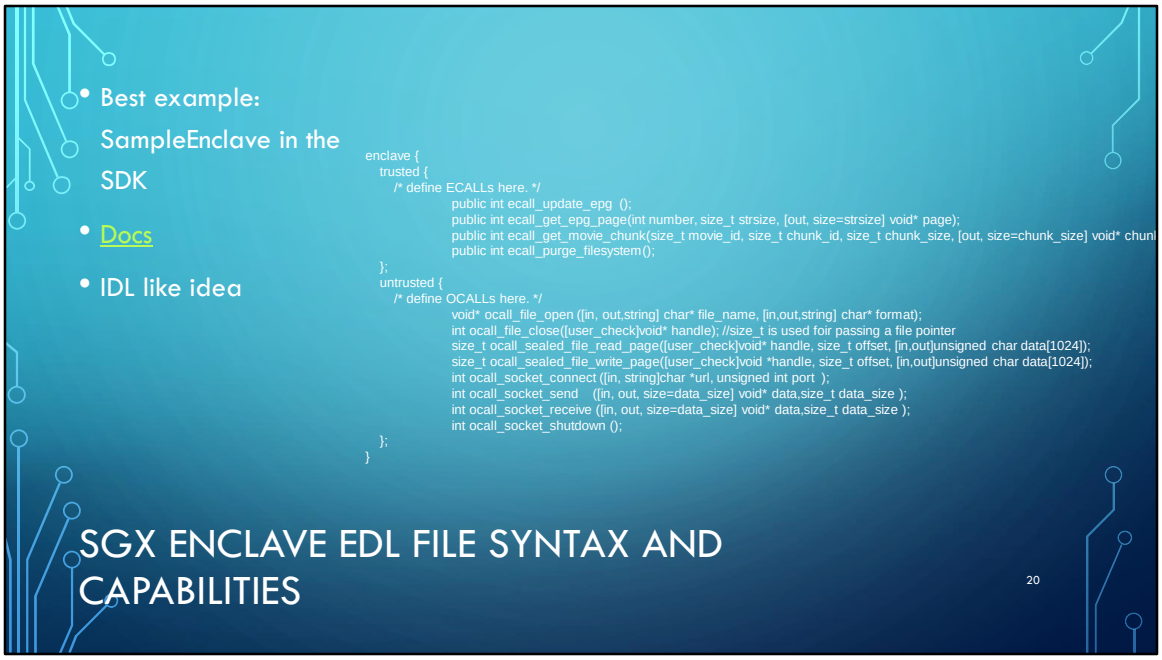
We have the environment to play with it both in Linux and Windows.
Here is windows environment definition:

Installation guide
Revision: 1.7 (Intel® SGX SDK version: 1.7.100.35600)
Visual Studio 2013/2015
Visual Studio 2013/2015 (mentioned as visual studio 2015 professional in the docs, I am using community edition and it works: the SDK files also has VS 2013 related files)
Debugger is included for debug mode enclaves
Enclave simulation exists
Required Hardware: ”6th generation Intel® Core™ processor (or later) based platform with Intel SGX-enabled BIOS support”



And here is a Linux environment definition, please remember to install both PSW and SDK on both targets.

Installation guide
Revision: 1.8 (Linux 1.8 Open Source)
Eclipse with the plugin
Debugger is included for debug mode enclaves (sgx_gdb utility)
Required Hardware: ”6th generation Intel® Core™ processor (or later) based platform with Intel SGX-enabled BIOS support”



Code that can not interact with user and/or other code is invisible, inaudible, unobservable, can not do anything and can be considered non-existing.
Enclave, as it was said before, should have well defined interfaces and here is the way in which these interfaces are defined:
EDL file (it looks like that edl means enclave definition language).
The idea behind it is very similar to the COM interfaces definitions: there are some definition of the call inside and outside of an enclave.
Generally it is a rich language – there are imports, complex data types, etc.

SGX ENCLAVE EDL FILE SYNTAX AND CAPABILITIES

- Best example: SampleEnclave in the SDK
- Docs
- IDL like idea

```
enclave {
    trusted {
        /* define ECALLs here. */
        public int ecall_update_epg_();
        public int ecall_get_epg_page(int number, size_t strsize, [out, size=strsize] void* page);
        public int ecall_get_movie_chunk(size_t movie_id, size_t chunk_id, size_t chunk_size, [out, size=chunk_size] vo
        public int ecall_purge_filesystem();
    };
    untrusted {
        /* define OCALLs here. */
                void* ocall_file_open ([in, out,string] char* file_name, [in,out,string] char* format);
                int ocall_file_close([user_check]void* handle); //size_t is used foir passing a file pointer
                size_t ocall_sealed_file_read_page([user_check]void* handle, size_t offset, [in,out]unsigned char data[1024]);
                size_t ocall_sealed_file_write_page([user_check]void *handle, size_t offset, [in,out]unsigned char data[1024]);
                int ocall_socket_connect ([in, string]char *url, unsigned int port );
                int ocall_socket_send    ([in, out, size=data_size] void* data,size_t data_size );
                int ocall_socket_receive ([in, out, size=data_size] void* data,size_t data_size );
                int ocall_socket_shutdown ();
    };
}
```

21

In this picture trusted section means calls into the enclave, named ECALLS

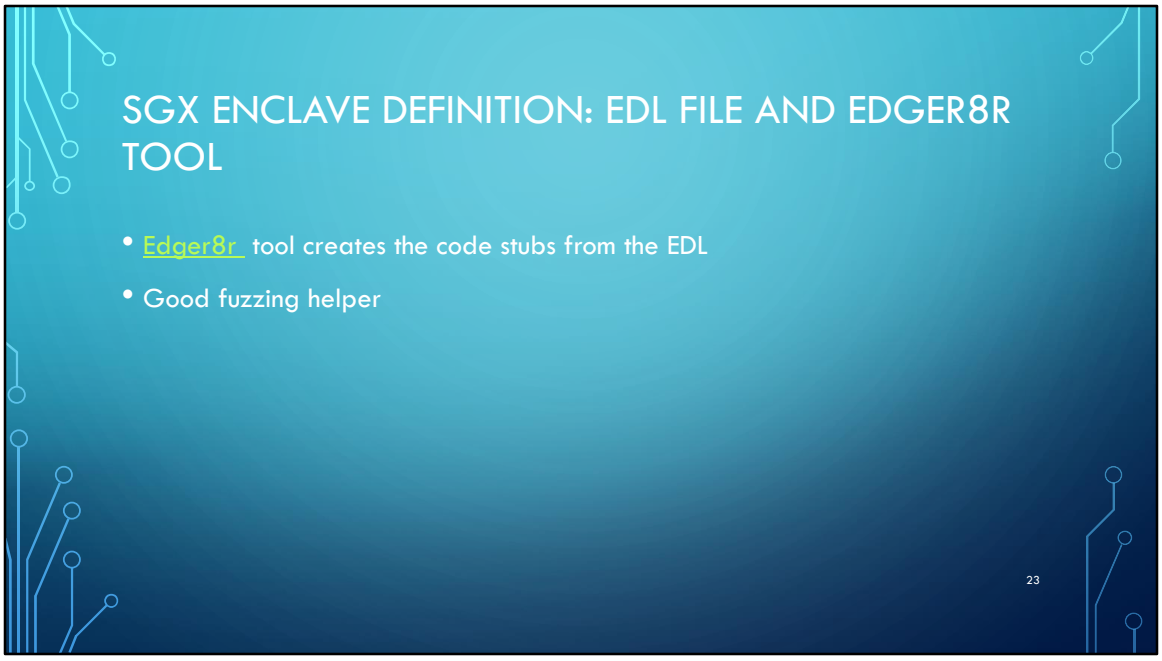
SGX ENCLAVE EDL FILE SYNTAX AND CAPABILITIES

- Best example: SampleEnclave in the SDK
- Docs
- IDL like idea

```
enclave {
    trusted {
        /* define ECALLs here. */
                public int ecall_update_epg  ();
                public int ecall_get_epg_page(int number, size_t strsize, [out, size=strsize] void* page);
                public int ecall_get_movie_chunk(size_t movie_id, size_t chunk_id, size_t chunk_size, [out, size=chunk_size] void* chunk
                public int ecall_purge_filesystem();
    };
    untrusted {
        /* define OCALLs here. */
                void* ocall_file_open ([in, out,string] char* file_name, [in,out,string] char* format);
                int ocall_file_close([user_check]void* handle); //size_t is used foir passing a file pointer
                size_t ocall_sealed_file_read_page([user_check]void* handle, size_t offset, [in,out]unsigned char data[1024]);
                size_t ocall_sealed_file_write_page([user_check]void *handle, size_t offset, [in,out]unsigned char data[1024]);
                int ocall_socket_connect ([in, string]char *url, unsigned int port  );
                int ocall_socket_send    ([in, out, size=data_size] void* data,size_t data_size );
                int ocall_socket_receive ([in, out, size=data_size] void* data,size_t data_size );
                int ocall_socket_shutdown ();
    };
}
```

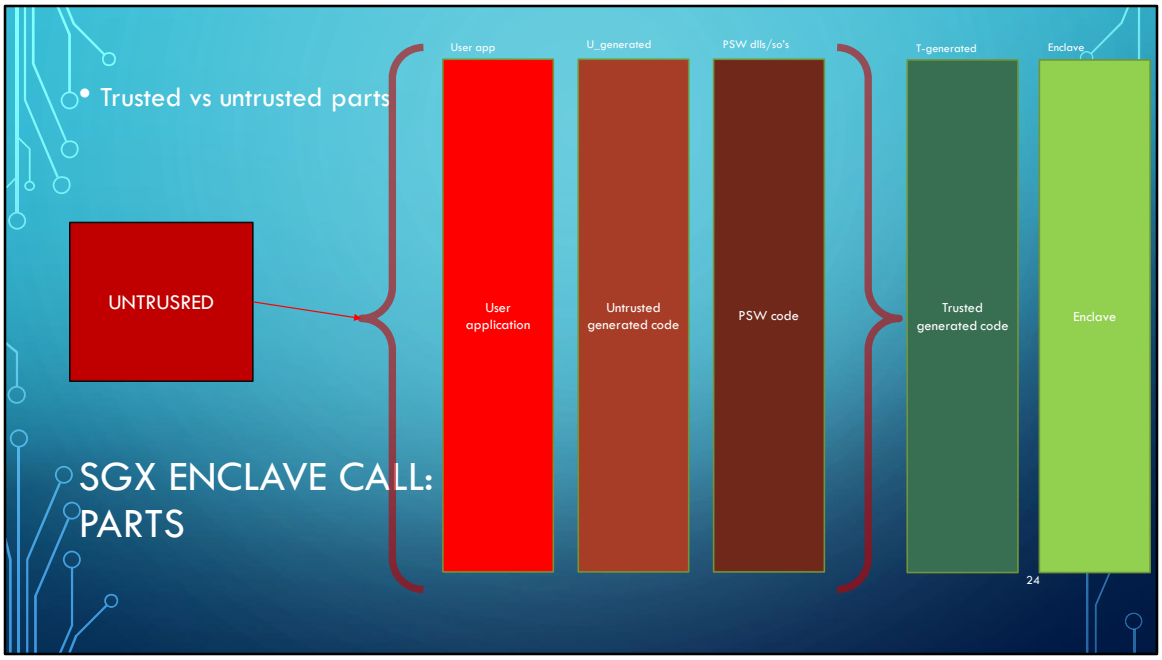
22

And untrusted – calls out of the enclave to untrusted environment (OCALLS)
As you remember the enclave works with ring 3 privileges and can not execute system calls:
Ocalls are intended to provide this service, but it requires leaving the enclave.

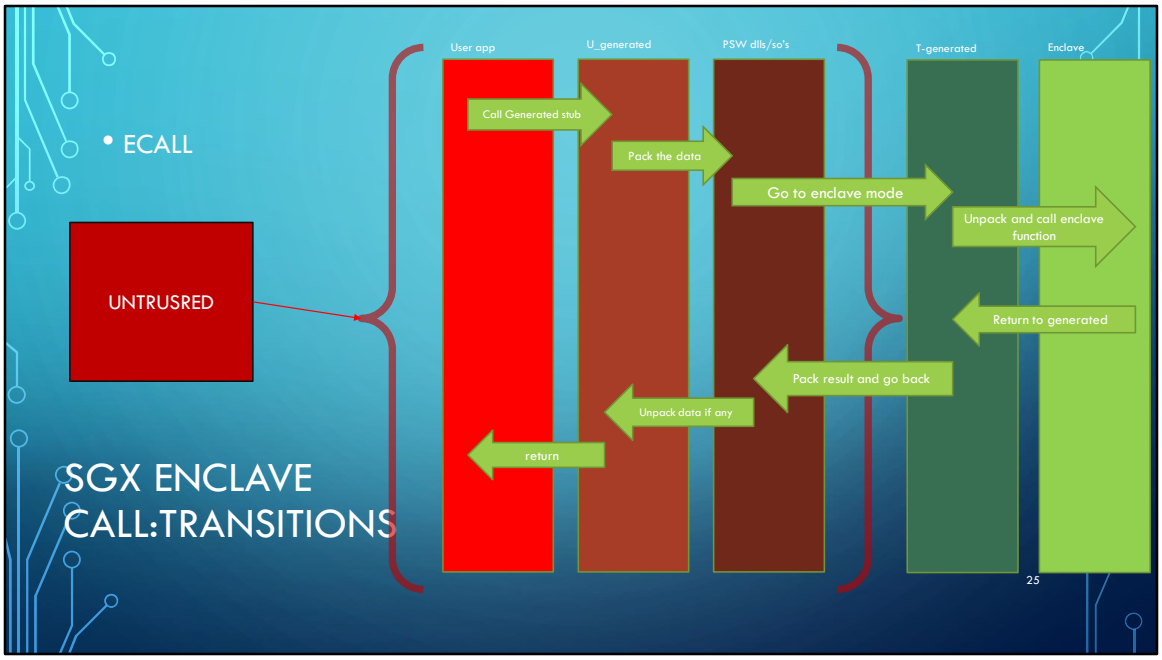


The EDL file is not compiled directly, there is some utility that generates a code from it in order to push some data inside an enclave and get some data out of it.
This is done with tool called edger8r – it generates code in C to solve this problem. This code is added automatically into the project (both enclave and enclave driving application for trusted and untrusted parts correspondingly) by visual studio SGX plugin.

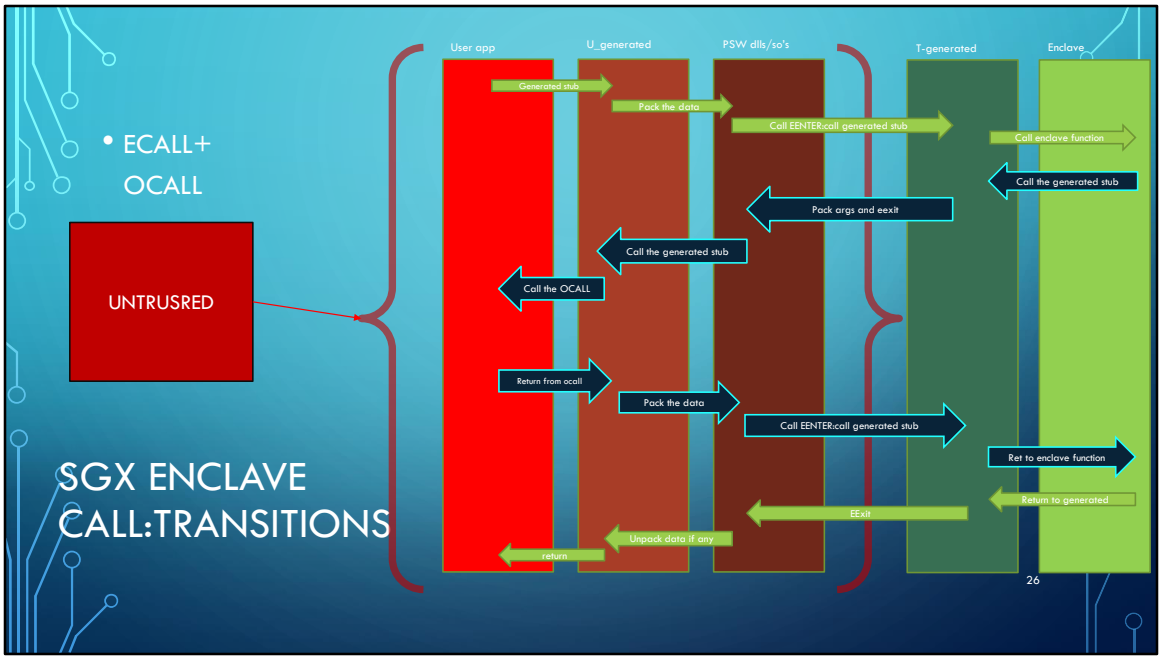
By the way, it is a great tool for building enclave fuzzing harness once you have or can deduce EDLs file content correctly(just import it to the project from the enclave or elsewhere in Visual Studio)



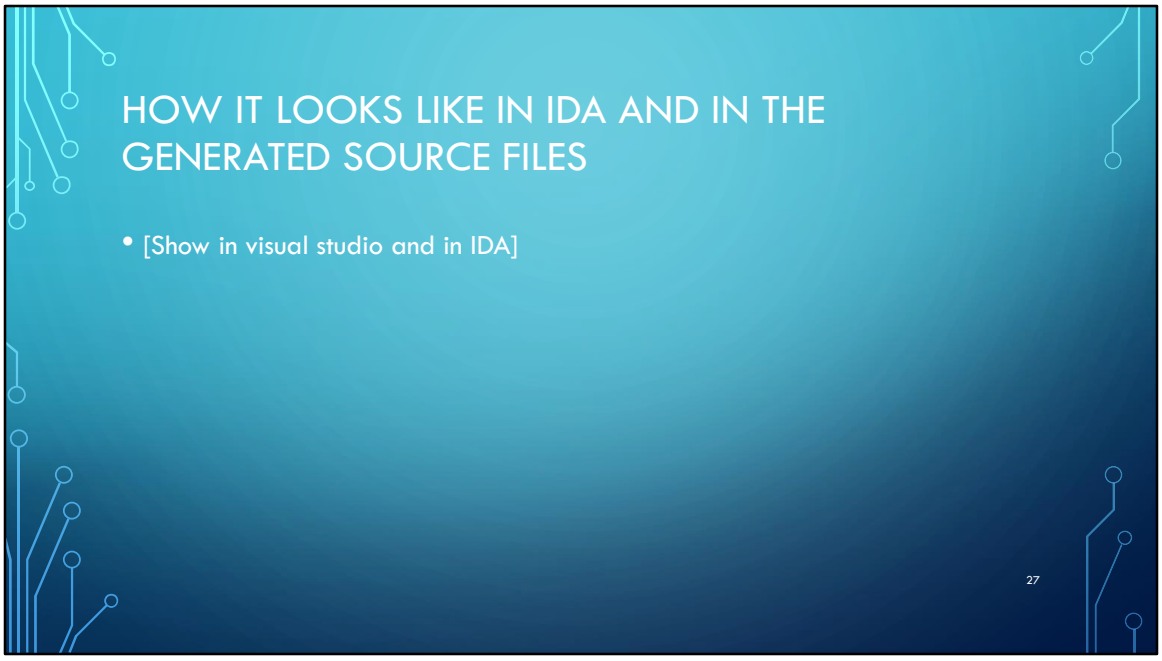
Let's talk about how these calls are implemented: we have the following parts: user application, untrusted generated code, Platform software code, trusted generated code, and trusted enclave code.
Blocks looking similar to red are untrusted and running in standard hackable mode.
Blocks looking similar to green are trusted and running in enclave mode.



Let's try to imagine an e-call and understand how it works from trusted/untrusted/generated/written manually code.
Here is how all transitions between the code parts looks like …



Let's add OCALL to this picture: (call to untrusted service such as open file, which should require transfer from ring3 to ring 0), it makes things a bit more complicated

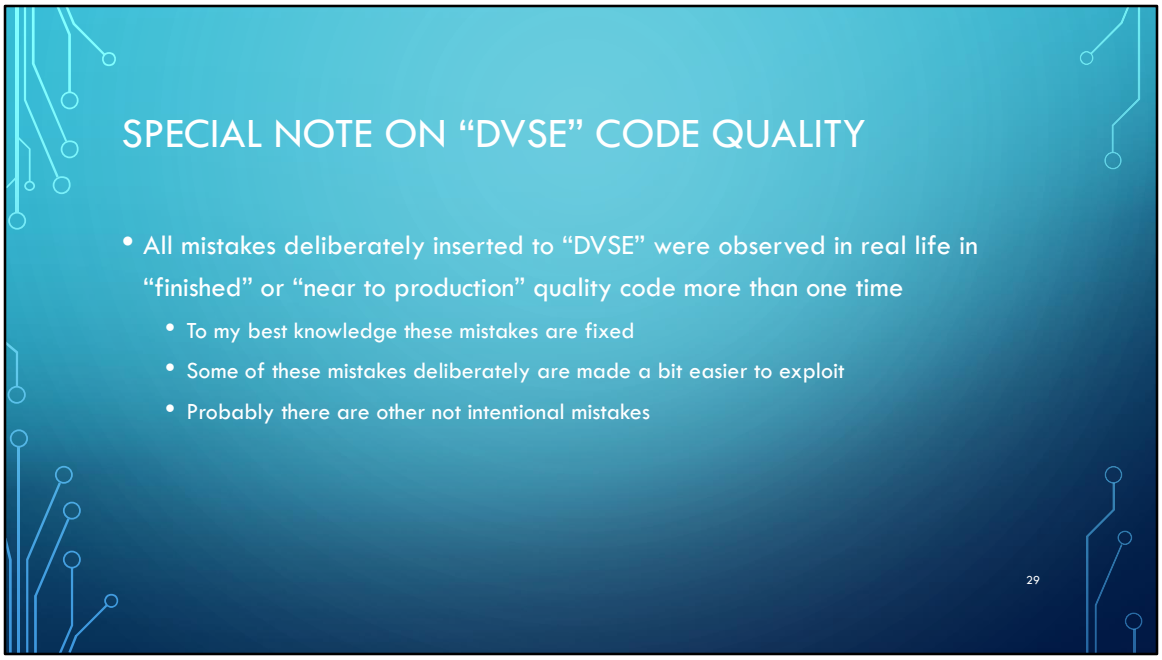


And now let's look how all these transitions are look like in the code.

Show the call from user application, generated code, enclu in Visual studio
Show cross-references to ecall in IDA

Now we have seen how it should work, let's start with overviewing the promised DVSE
First of all – the link is here (mention if it is public for now):

You can (and should) build it and debug it as you can and as platform allows
Debugging flag is a part of the key derivation material – which makes release build much more interesting target
If you want to use this in release environment you have to sign it yourself on behalf of yourself or your organization
Unfortunately I can not distribute signed enclave because of licensing issues.



There are some statements I have to state loud and clear, but this time not for the legal gods sake:

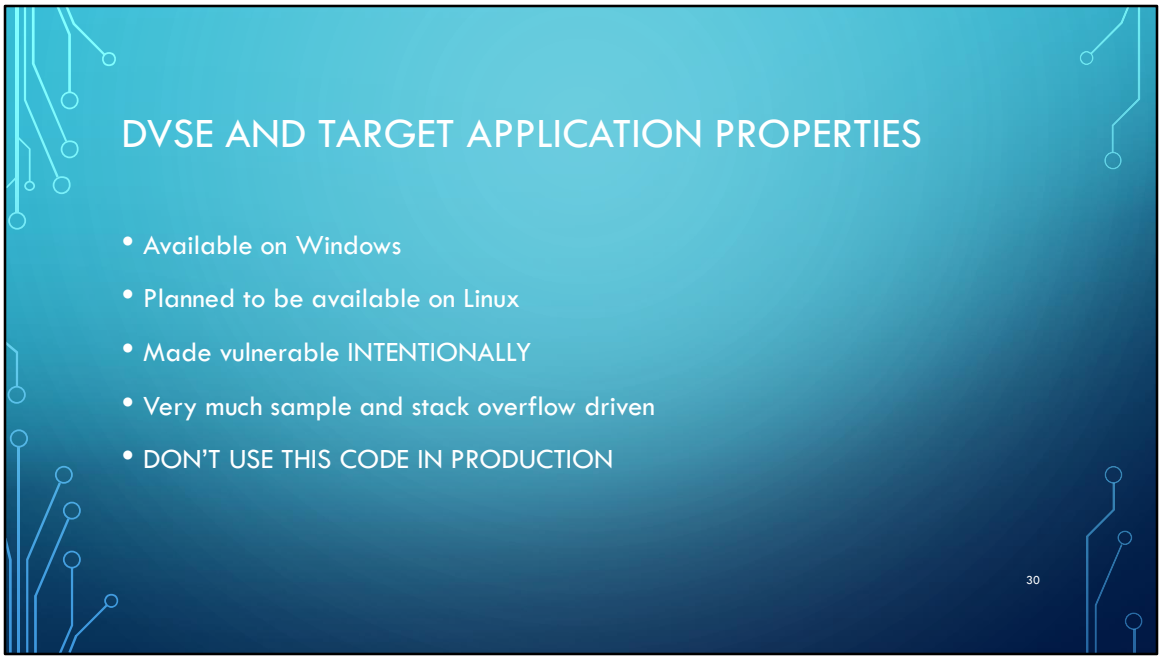
All mistakes I deliberately inserted to “DVSE” were observed in real life in the code which was marked as “finished” or “near to production” quality

To my best knowledge all mistakes I am going to speak about were fixed in original enclaves (but I saw these mistakes enough times to make conclusion that these mistakes are common)

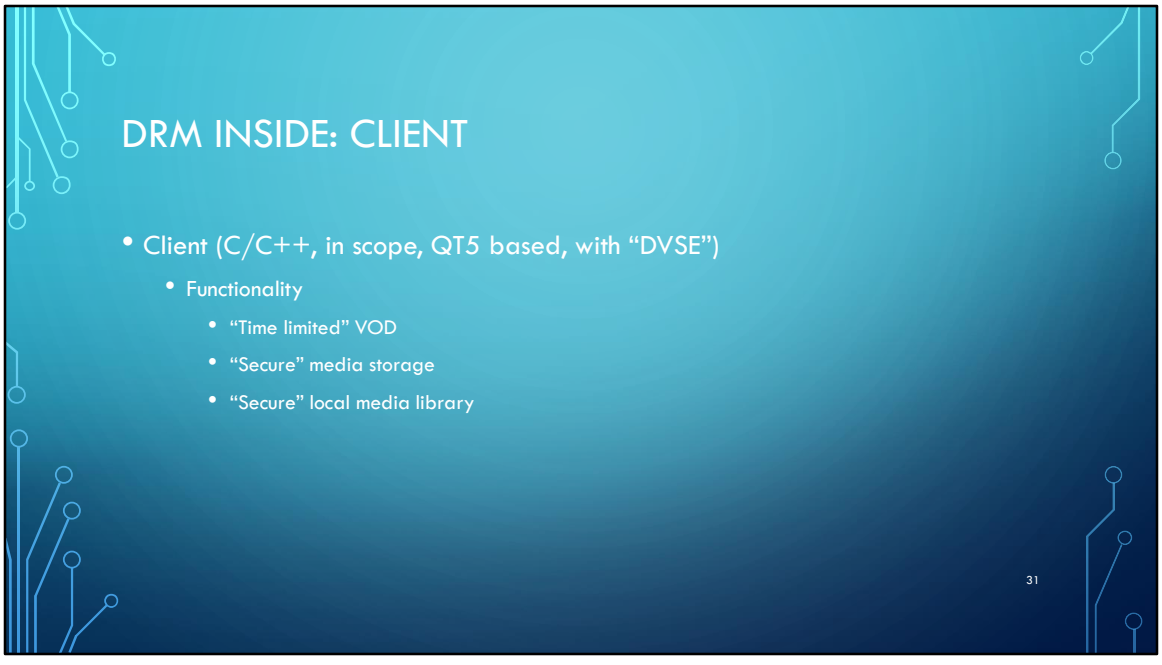
Some of these mistakes deliberately are made a bit easier to exploit

It doesn't mean, however, that there are no other mistakes I made not intentionally

In order to write the DVSE I had to resurrect the bad programmer buried inside of myself since I stopped to work as such.
It was very hard. In addition it was very interesting psychological experience.
I even had an idea to publish it on behalf of imaginary programmer with complicated name taken from Russian literature such as Васисуалий Лоханкин – but abandoned it.

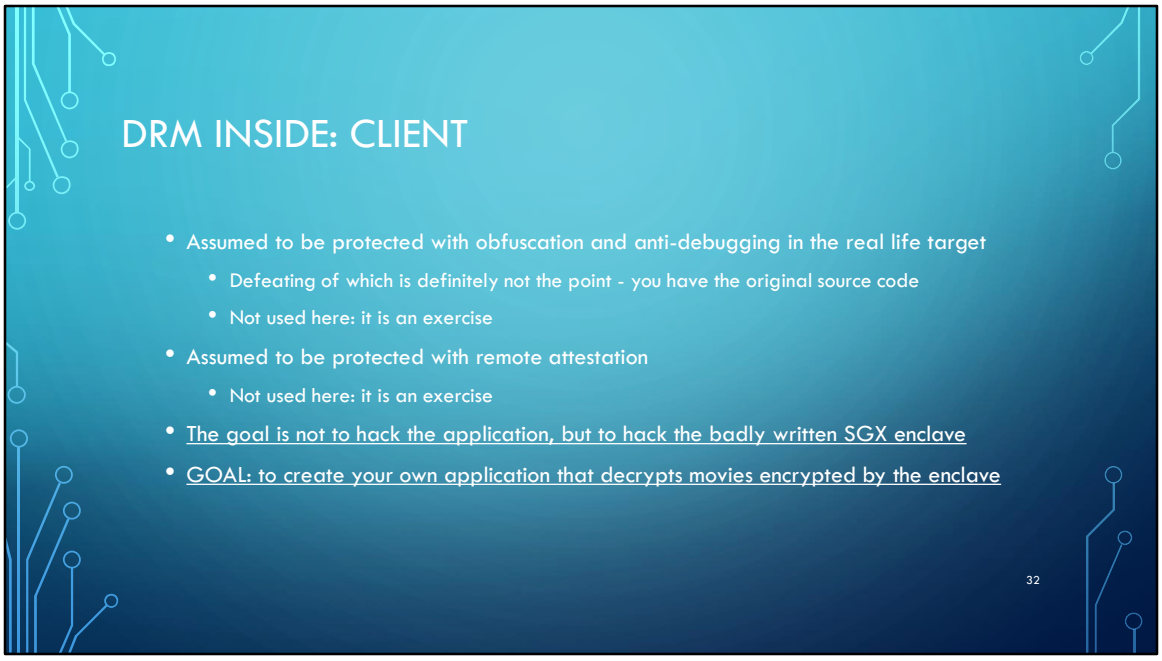


So, what is it? On what exactly are we training ?
This is windows application I planning to port on Linux sometimes,
Intentionally vulnerable, partially copylefted from stack-overflow and samples,
And of course, completely not usable in production.
I beg you. Please don't shoot yourself in the leg, you were warned.

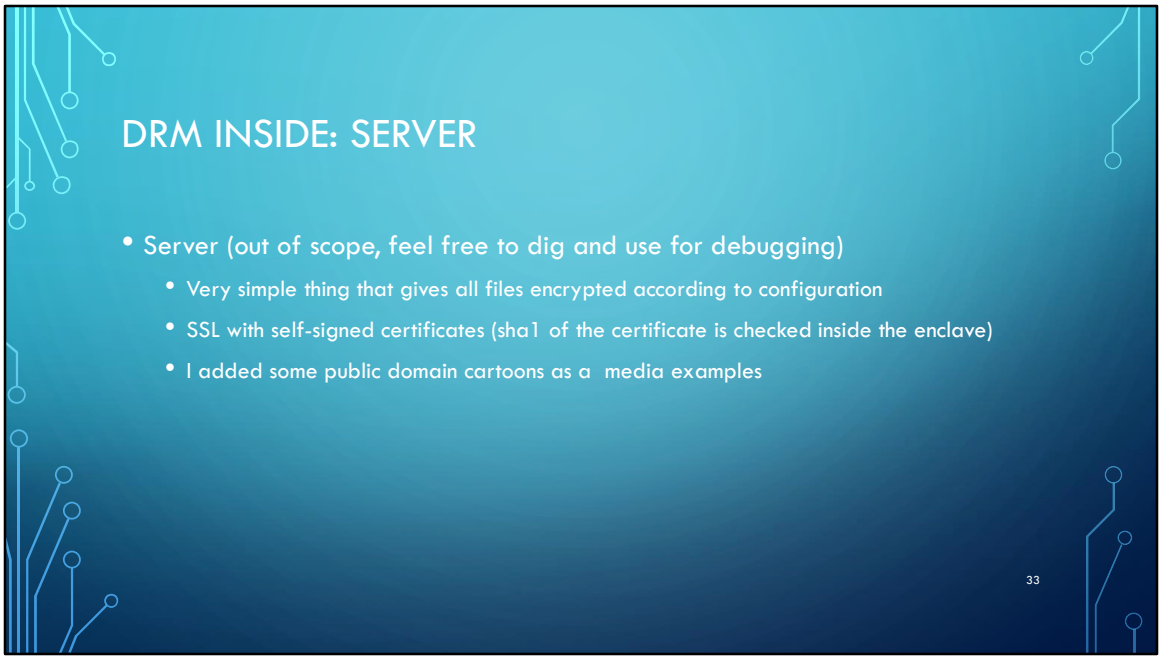


The application consists of client, server, and the evil DRM.
This evil DRM includes:

"Time limited" VOD
"Secure" media storage
"Secure" local library



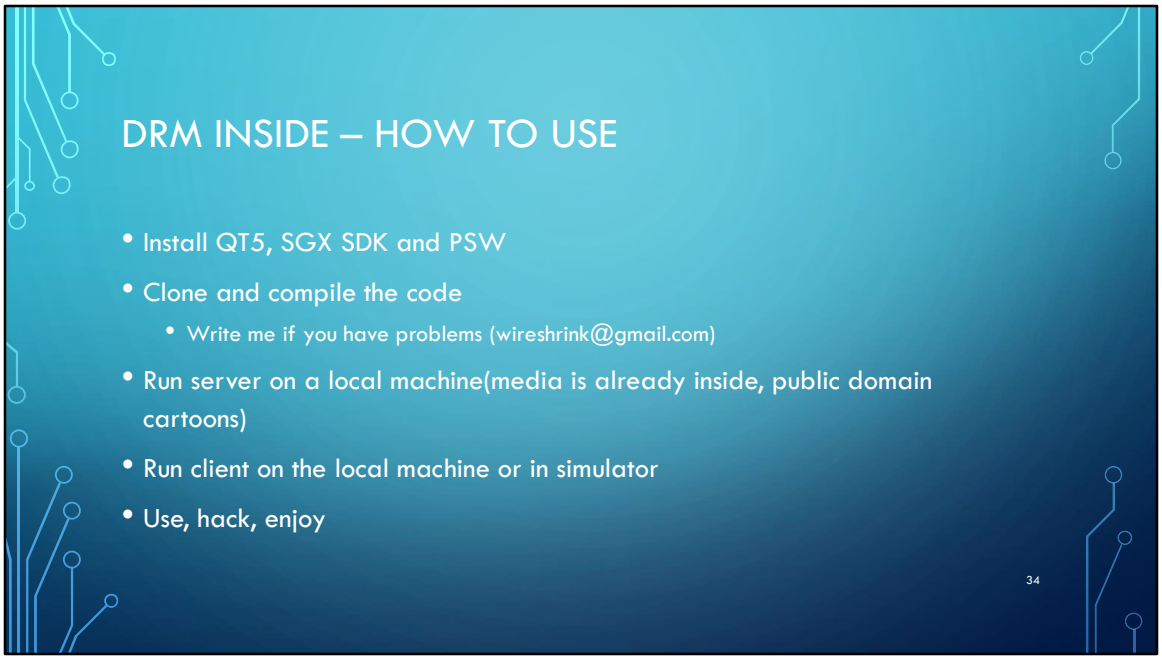
In the real life this kind of applications will be protected with obfuscation, remote attestation and other tricks in untrusted part. I decided to leave application protection out of scope –
Defeating it is not the point.
The main goal of this exercise is to create another application that will get all the enclave's secrets and will be able to decrypt the movies.



Server gives up all the files if asked correctly, and defeating it is also not the point: Server is out of scope too.

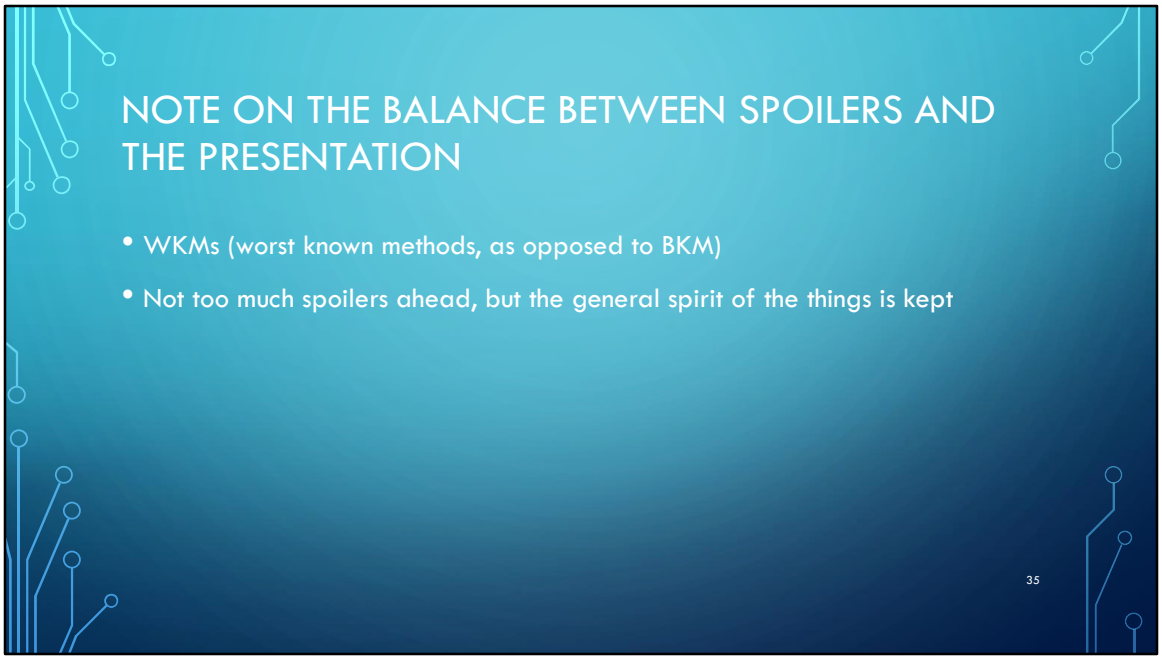
It is

- Very simple python script thing that gives all files encrypted according to configuration
- SSL with self-signed certificates (sha1 of the certificate is checked inside the enclave)
- I added some public domain cartoons as a media examples (At least I downloaded these cartoons from the internet site with this claim)



So here you can see a general usage instructions: install components, compile the code including the enclave, run server and client on the local machine as wrote in readme.md.

In short – use, hack, enjoy.

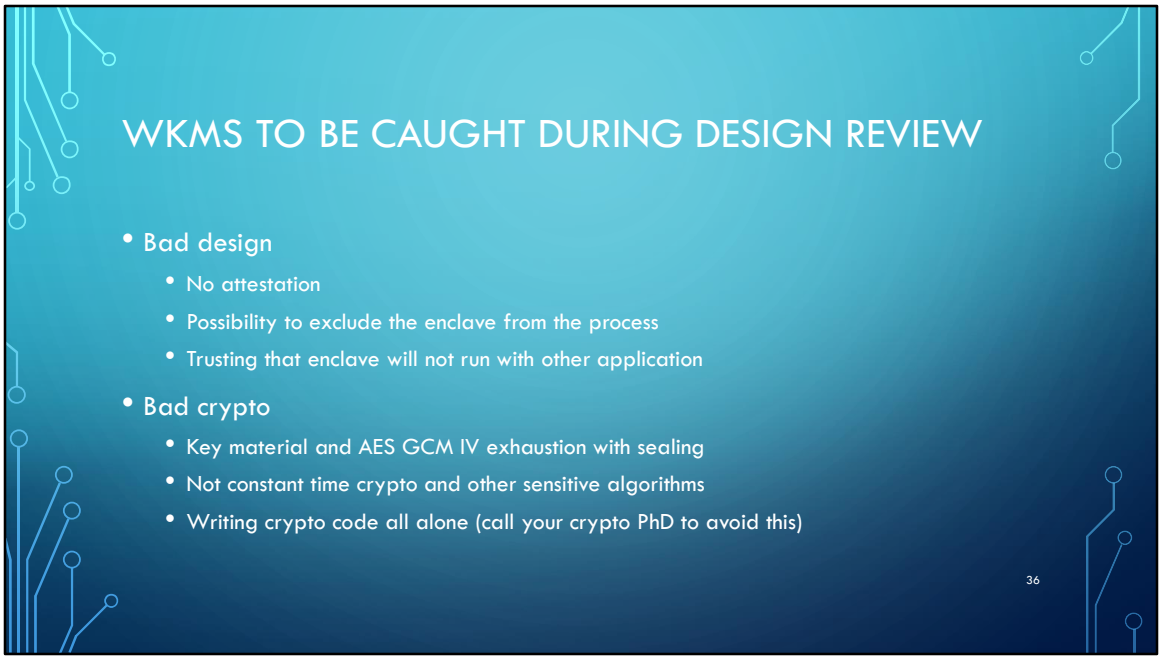


Then question is – what mistakes are there. Now we going to explore the promised WKMs I observed on real–life enclaves.

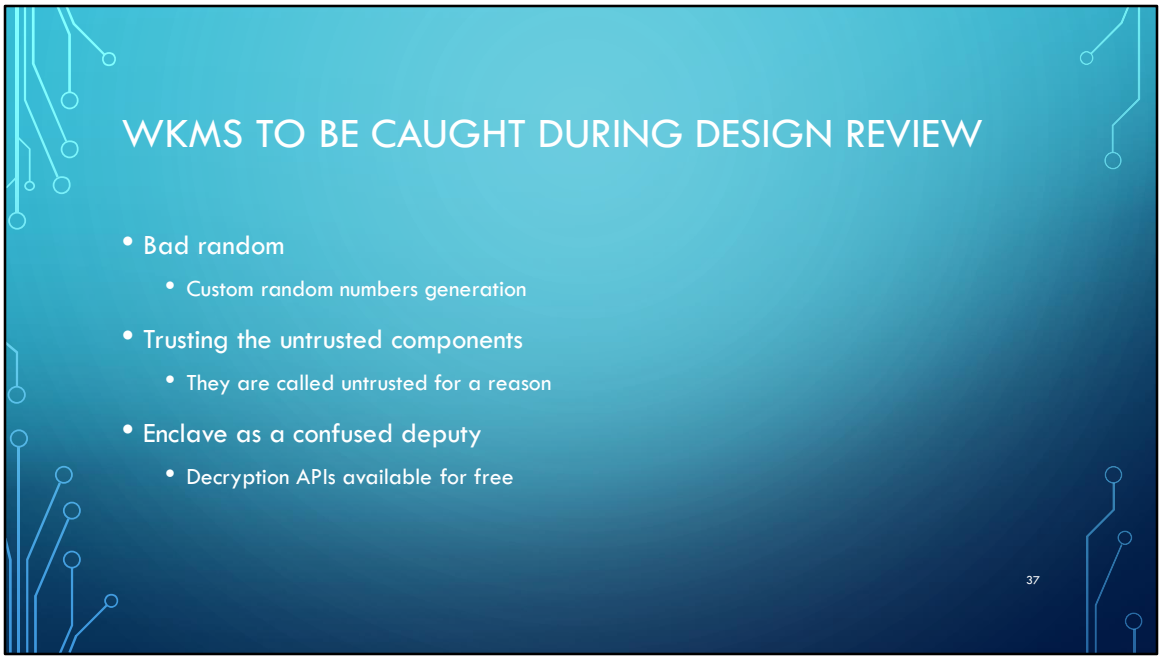
System programming provides a lot of possibilities to shoot yourself in the leg. SGX enclave programming is not an exception, and we going to explore these possibilities just now.
We speaking about:

WKMs (worst known methods, as opposed to BKM)

Please do not expect too much implementation spoilers ahead, but the general spirit of the things is kept

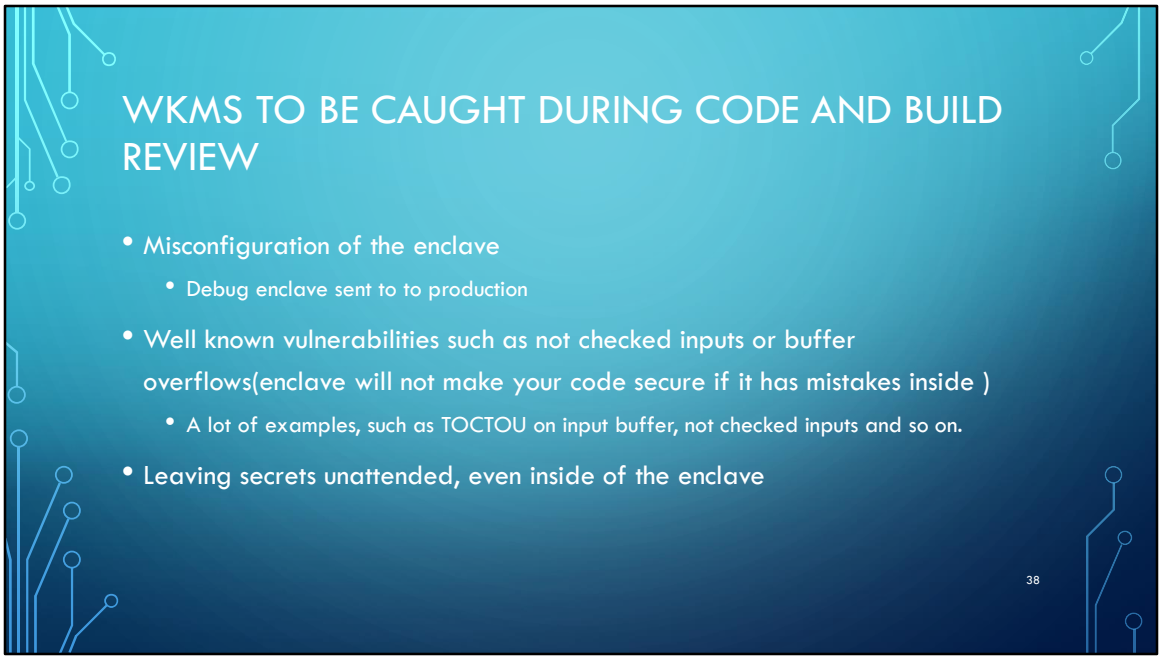


I divided these WKMs according to the development phase they should be discovered.
So, design review should avoid – surprise – bad design :)



Please verify every single line of the code, bad random is very hard to catch.
I understand the noble idea “we can not audit rdrand instruction and then we’ll use it only for random number seeding”
After taking this unfortunate decision the responsibility on this is yours alone.

Decrypt secret using the hidden key shouldn’t be an ECALL if enclave doesn’t check outside environment, which is almost impossible to do reliably



Code and build review can reveal the following things

Misconfiguration of the enclave

Debug enclave sent to production

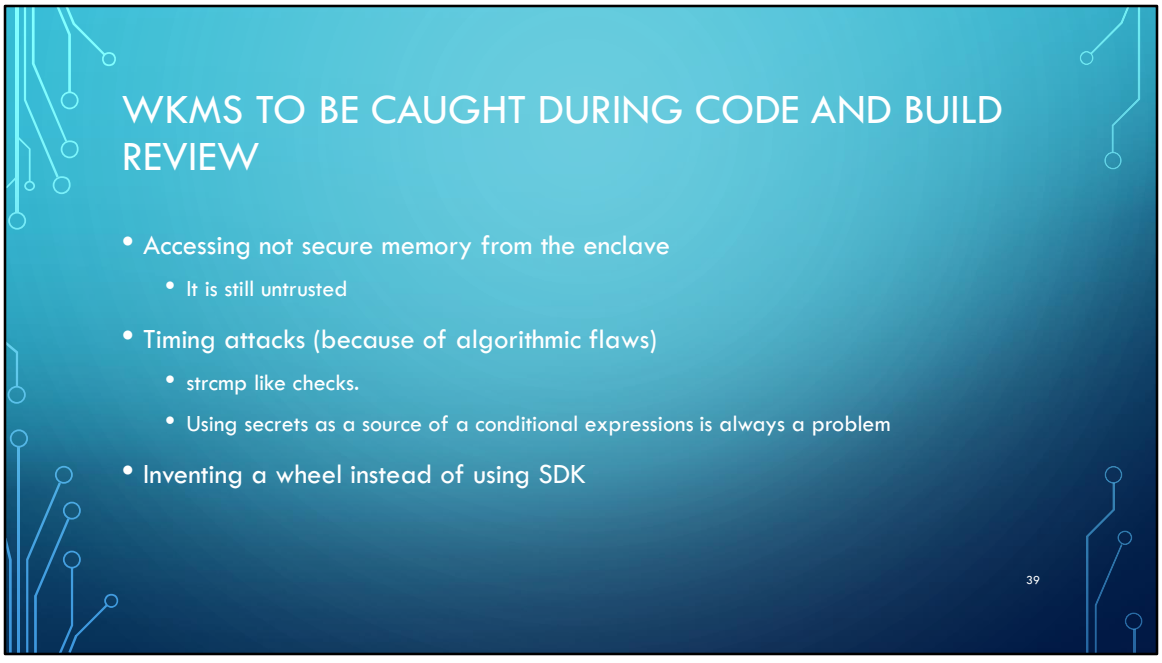
Well known vulnerabilities such as not checked inputs or buffer overflows(enclave will not make your code secure if it has mistakes inside)

A lot of examples, such as TOCTOU on input buffer, not checked inputs and so on.

Leaving secrets unattended, even inside of the enclave

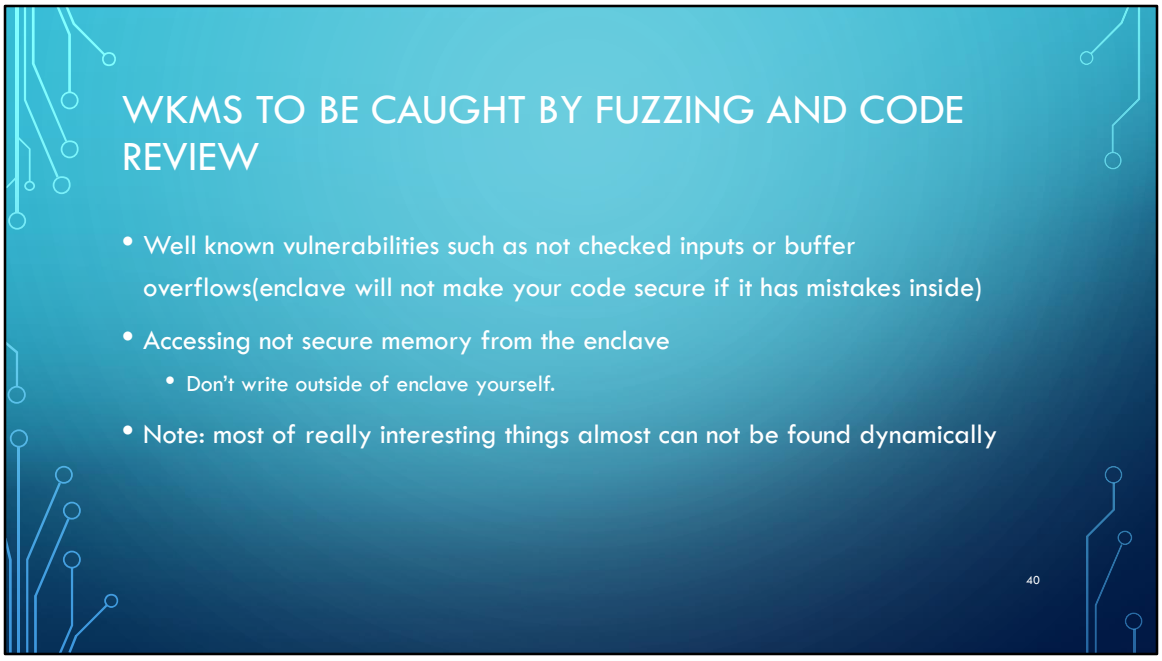
Use memset_s to clear secrets

Clear secrets as soon as possible after usage

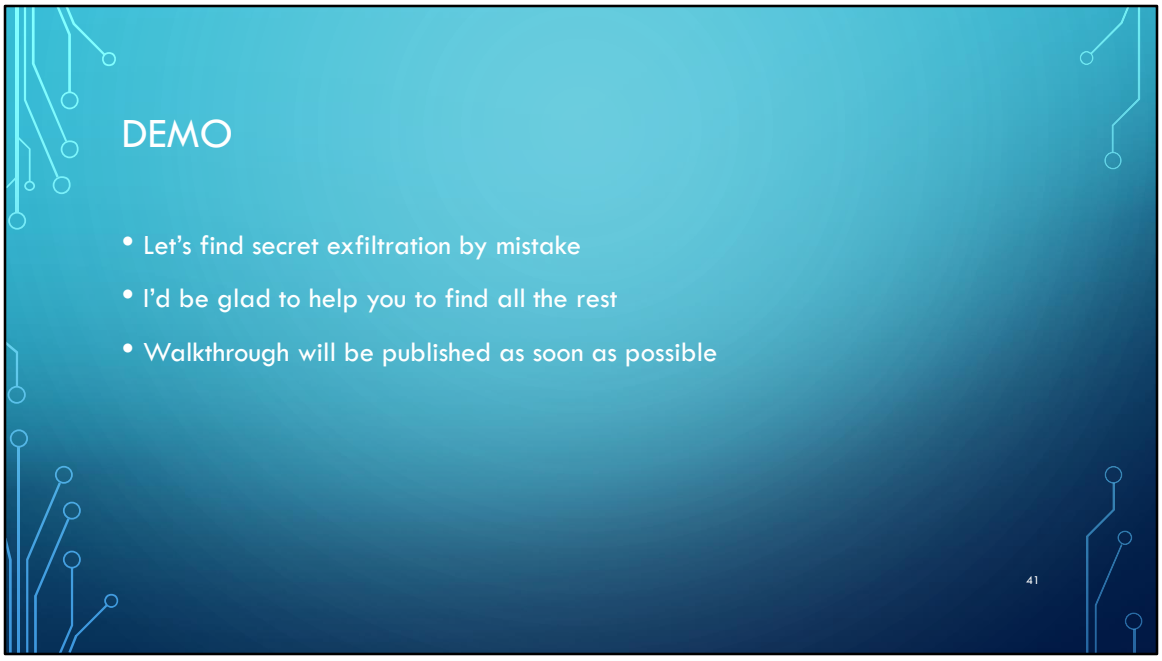


More things that should be caught there:

Accessing not secure memory from the enclave
    It is still untrusted
Timing attacks (because of algorithmic flaws)
    strcmp like checks.
    Using secrets as a source of a conditional expressions is always a problem
Inventing a wheel instead of using SDK
    Please don't implement sealing, multithreading, crypto, and other things existing in the SDK yourself.
    I'd seen some of custom implementations of this idea, none of them made sense, and all of them ended badly.



And fuzzing, in the end.
Please note that most interesting things almost can not be found dynamically.
During the conference somebody mentioned "Bad design guard" – so wee really need it ☺



So, here is the demo – let's exfiltrate something from the enclave.
All the rest you'll have to either find yourself, or contact me for the solution or wait until it will be published.

Intended vulnerability – not checked index of the read buffer which leads to the data exfiltration (which was stored in the enclave for all the enclave life time)
This vulnerability is easy to find manually, and I'll show it to you.
1 – Run server, show config files(epg and coupons)
2 – Run client
3 – Mention coupons. Add library folder and init user
4 – Show the EPG
5 – Run the free cartoon and stop

6 – Show where EPG is got in Visual Studio (app) and IDA (enclave)
7 – Read through the code of the DVSE
8 – Show the memcpy
9 – Show the enclave test
10 – Show (show shortcuts: we using for the same EDL and even the same class for ECALLS, but the OCALLS are changed). Show generated files for Untrusted part.

11 - Run it
12 – Show found coupons
13 – Show certificate details

Note: revealing this vulnerability is easy, but it illustrates the following WKMs:

1 – leaving secrets unattended and thus very bad design (secrets should be cleared immediately after usage, just for defense in depth – and enclave didn't help us in anything). Here you have a lot of secrets.
2 – Standard vulnerabilities (and it can be found by fuzzing)

This vulnerability probably looks a bit artificial, but I have a really good reason for showing it:

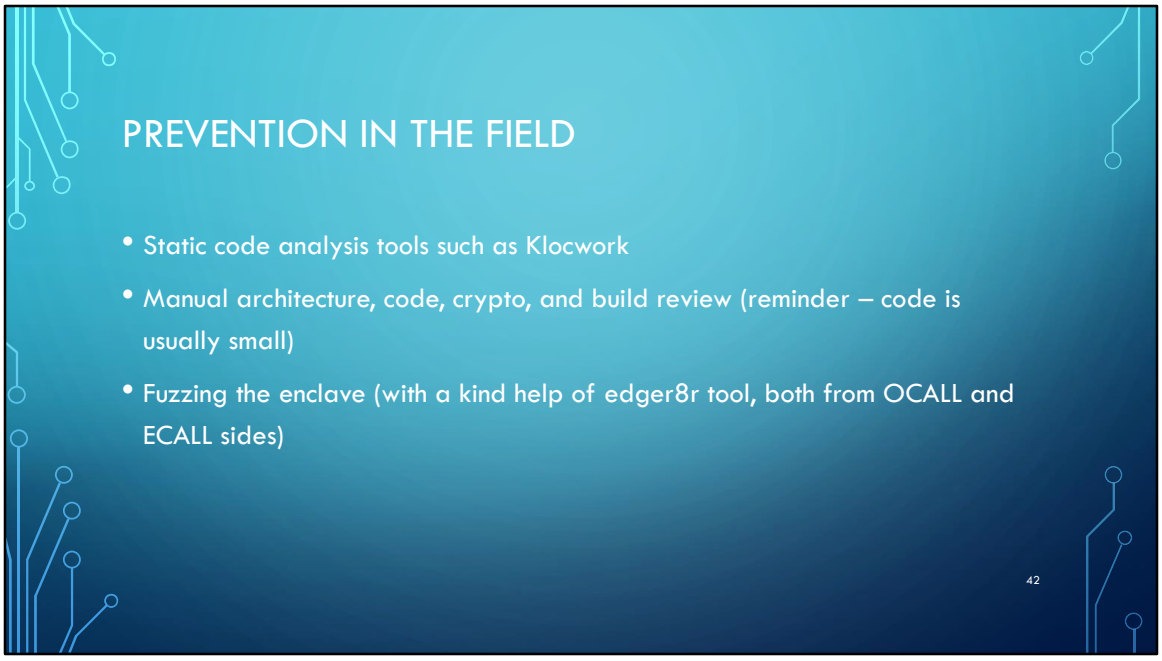
1 – As an example of how small and harmless issue can undermine all the security model

There is a proverb in Russian, "don't put all the eggs to the same basket", with very clear meaning: don't place all of your valuables at the same place.

In the boundaries of this analogy any kind of secure enclave, and it doesn't matter how exactly it is implemented, is a basket with eggs.

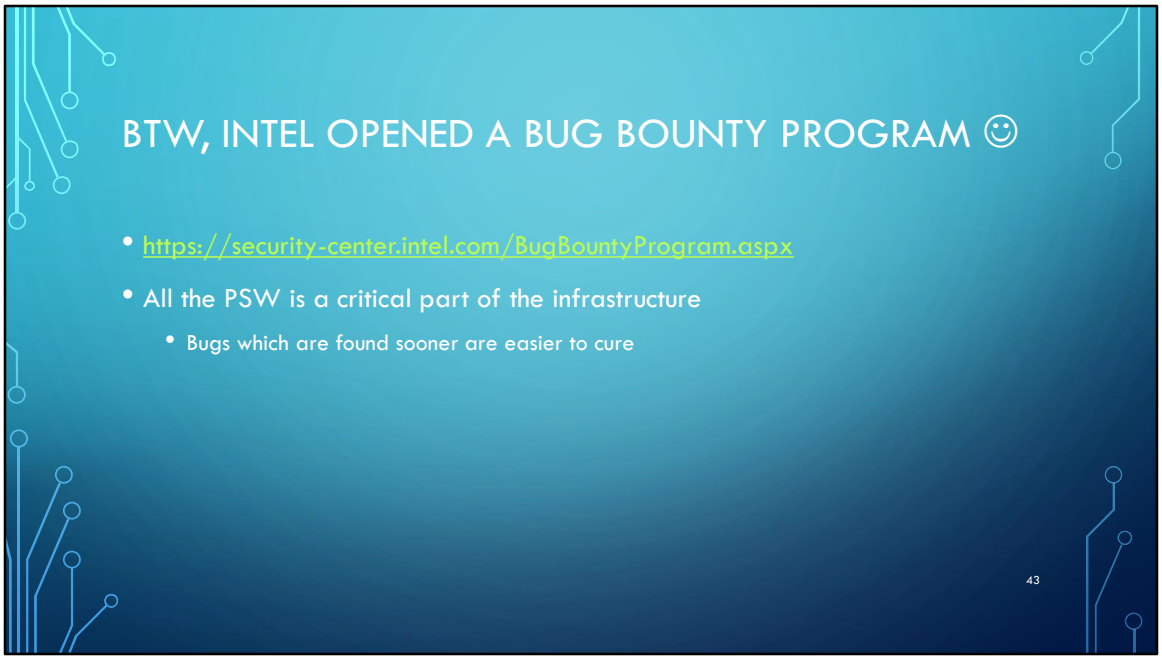
Once you drop it – you break everything.

2 – I had seen similar things in real life enclaves. They were a bit more complicated, but it is an exercise, after all.



Let's speak for a moment about prevention of these bugs.
Surprisingly automatic code analysis tools – such as Klocwork - may give a good result if used with nightly builds every day.
Manual review of all possible things gives best results – but with the results we have a problem:
- the review should be conducted by very much qualified people
- It is still probabilistic process, and something will always be overlooked
As for fuzzing -
Nothing new here except of old good and dumb enclave fuzzing – there are no tools that allow to use something more complicated.
The only hope here is that good enclave is small – and missing information can be obtained by code review.

Again "Bad design guard" to rescue … We definitely need it here.
In addition I think that symbolic execution may be interesting in this context, but I didn't use it myself for this purpose.



If you are reading these presenter notes – thank you very much ☺

FUTURE WORK (ORDER DOESN'T MATTER)

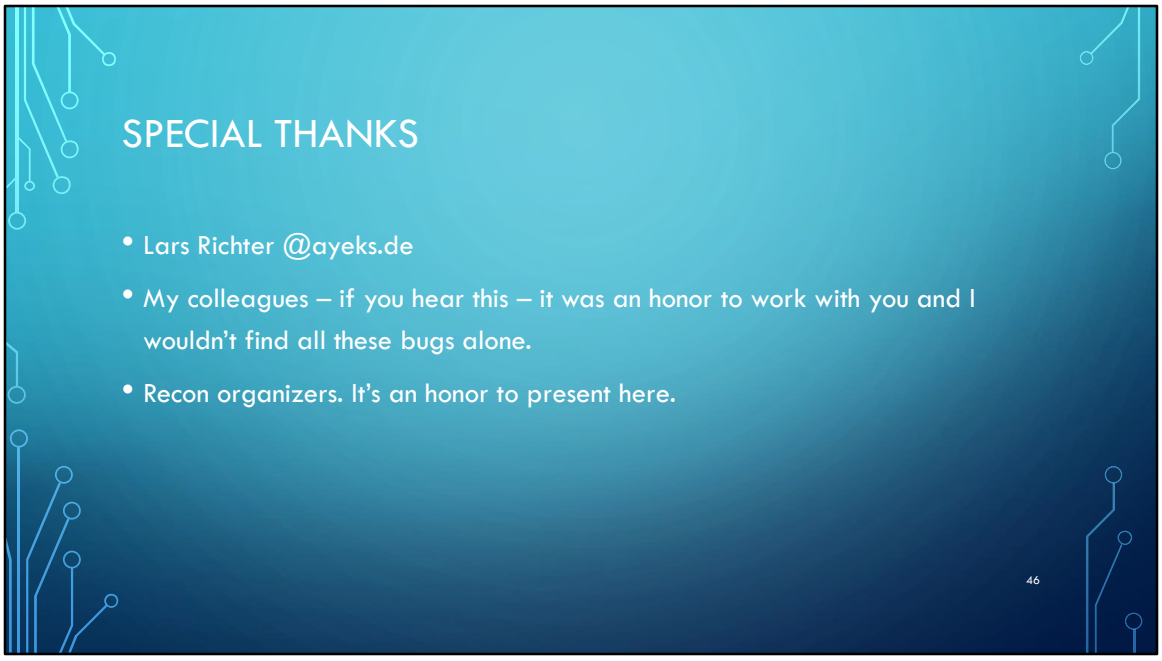
- Porting DVSE to Linux
- Publishing walkthrough and exploits
- Automatic deduction of the enclave ECALLs/OCALLs from binary analysis
- Abstract interpretation and/or symbolic execution for fuzzing (again, enclave is small)

ENCLAVE – USEFUL LINKS

- Where to start : https://software.intel.com/en-us/sgx
- ISCA 2015 SGX tutorial: https://software.intel.com/sites/default/files/332680-002.pdf
- Good external analysis: http://eprint.iacr.org/2016/086.pdf
- For HW - SDM is the best: Vol. 3D : http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf
- Linux SDK and platform software
  - https://01.org/intel-softwareguard-extensions
  - https://github.com/01org/linux-sgx
- Enclave writers guide https://software.intel.com/sites/default/files/managed/ae/48/Software-Guard-Extensions-Enclave-Writers-Guide.pdf

45

Some useful links for the concerned fellow citizens



Special thanks : as in list.

QUESTIONS

- Please ?

THANKS FOR WATCHING ☺

- Feel free to contact me
  - wireshrink@gmail.com
  - @wireshrink (I'm not writing there, but you can send PM, and news about DVSE will appear there)

BACKUP

NOTE ON SGX SEALING KEY DERIVATION MATERIAL

Table 40-56. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	MRENCLAVE	MRSIGNER	RAND
Source	Key Dependent Constant	Y← SECS.ATTRIBUTES and SECS.MISCSELECT;	CR_SGX OWNER EPOCH	Y← CPUSVN Register;	R← Req.ISVSVN;	SECS. ISVID	SECS. MRENCLAVE	SECS. MRSIGNER	Req. KEYID
		R←AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT;		R← Req.CPUSVN;					
EINITTOKEN	Yes	Request	Yes	Request	Request	Yes	No	Yes	Request
Report	Yes	Yes	Yes	Yes	No	No	Yes	No	Request
Seal	Yes	Request	Yes	Request	Request	Yes	Request	Request	Request
Provisioning	Yes	Request	No	Request	Request	Yes	No	Yes	Yes
Provisioning Seal	Yes	Request	No	Request	Request	Yes	No	Yes	Yes

- + Fuses, of course (CR_SEAL_FUSES in terms of SDM, the screenshot is from there)
- Note Sealing key – it is enclave or signer specific

50

One more note for history – key derivation material. This table actually shows that each enclave will have its own key, depending on a lot of parameters including debug state, enclave content or enclave certificate and fuzez

WHY CAN'T I SIGN IT MYSELF

- https://software.intel.com/en-us/license/intel-software-guard-extensions-licensee-guide

" In addition, Licensees should:

Observe industry secure coding best practices for software development to avoid vulnerabilities (such practices might include a secure software development framework, coding standards, data input validation, least access possible, secure logging, and so on).

Address and fix significant security vulnerabilities within a reasonable time, or within a time frame established under existing disclosure arrangements between Intel and the Licensee, after becoming aware of the vulnerability.

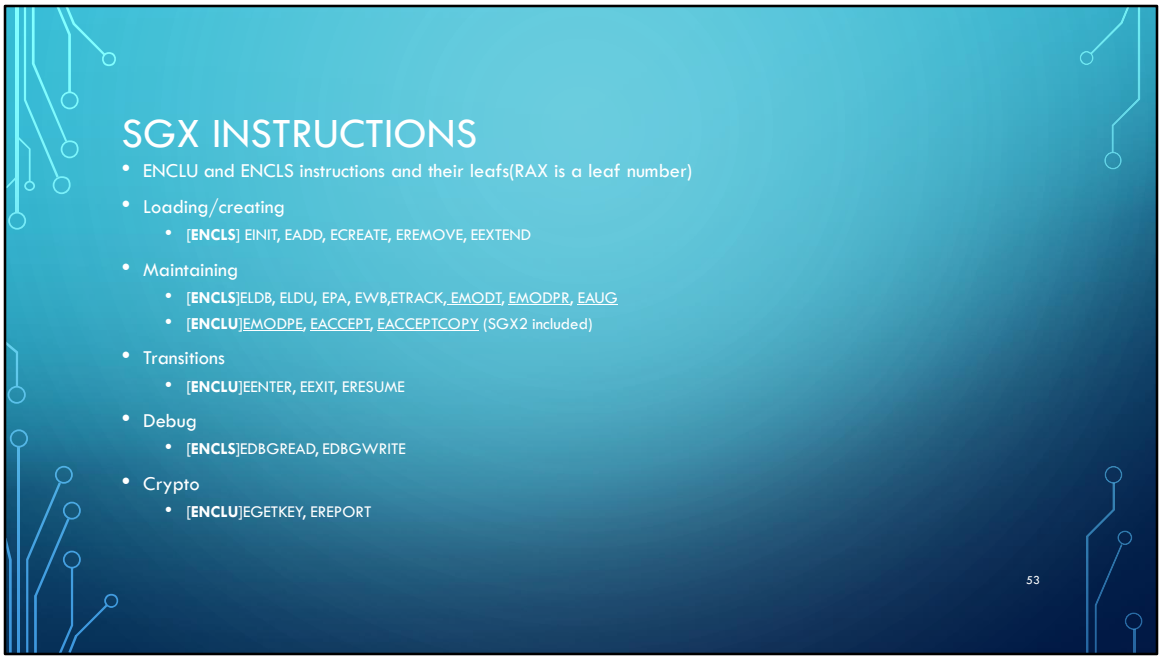
Ensure that the licensed application installer, or the operating environment in which the application resides, includes the most current Platform Software (PSW) Installer for Intel SGX.

Ensure that end-users receive PSW updates via application update mechanism, or via the operating environment in which the application resides.

Observe best industry practices to: (i) not write malware, spyware or other nuisance software; (ii) **not write poorly designed software that contains significant security vulnerabilities or that fails to deliver its security promise**.

Construct Licensed Software Applications to enable complete removal on end user request, including removal of any sealed data."

If we are not speaking about local/remote attestation – AES GCM is the crypto algorithm used in sealing the secrets by default (as in Linux SDK source code). EPID and other things (a lot of them) are used for the attestation, licensing and quoting: all of them are out of scope for this talk



Yes, we have extensions for extensions ☺