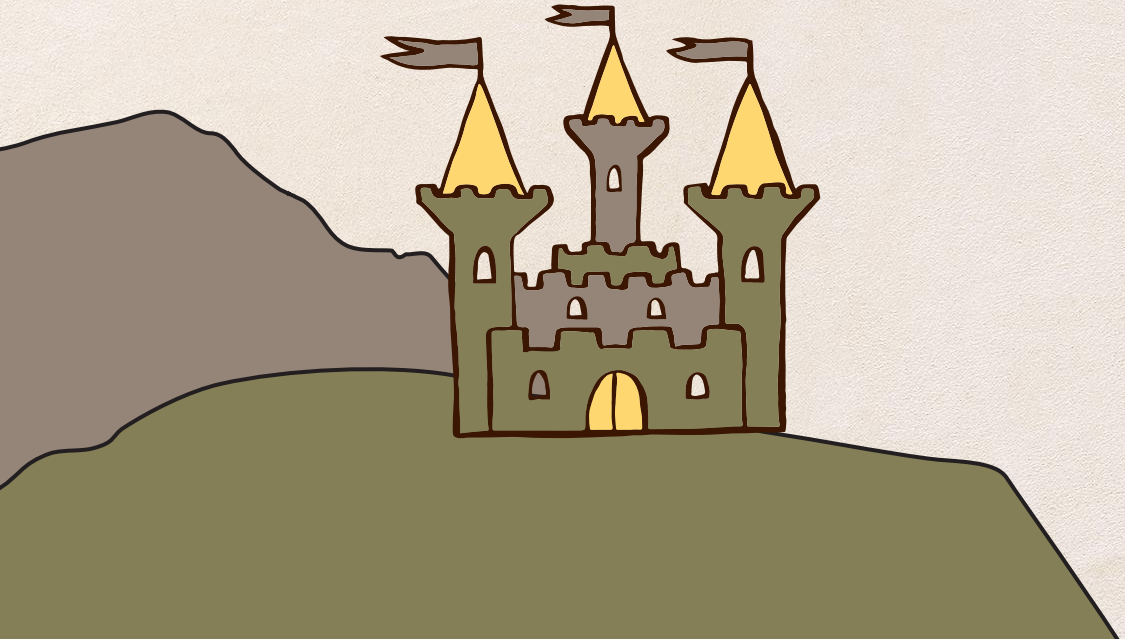
Cryptle: a secure multi-party Wordle clone with Enarx

By Tom Dohrmann, Richard Zak, and Nick Vidal

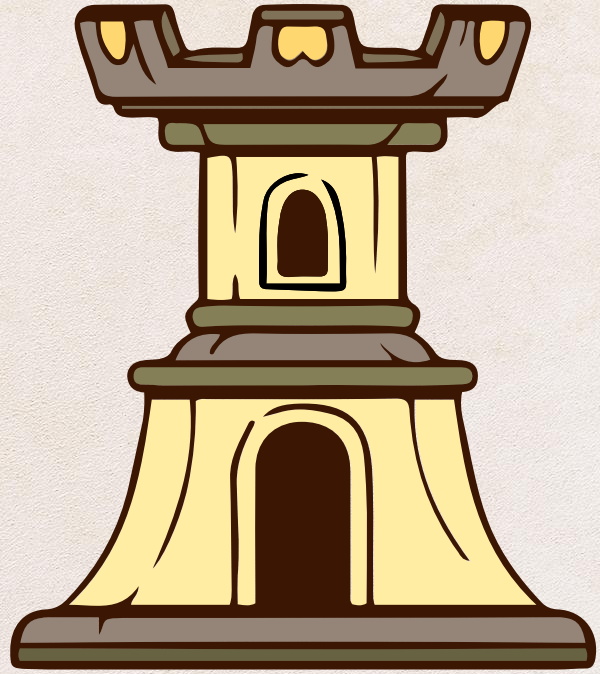
Table of contents



01

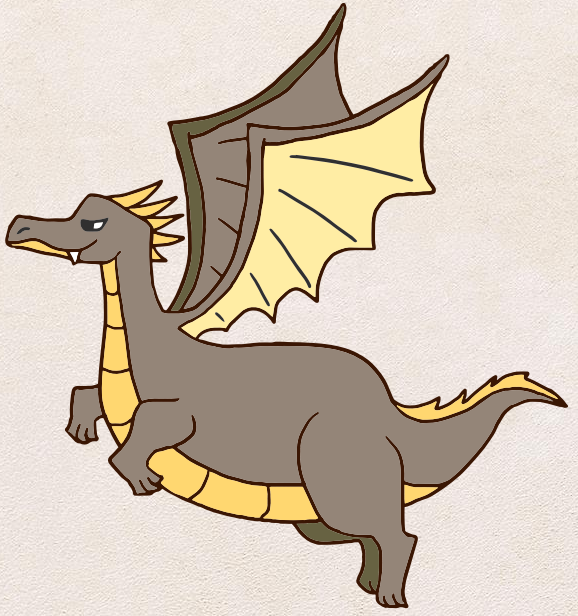
Confidential Computing

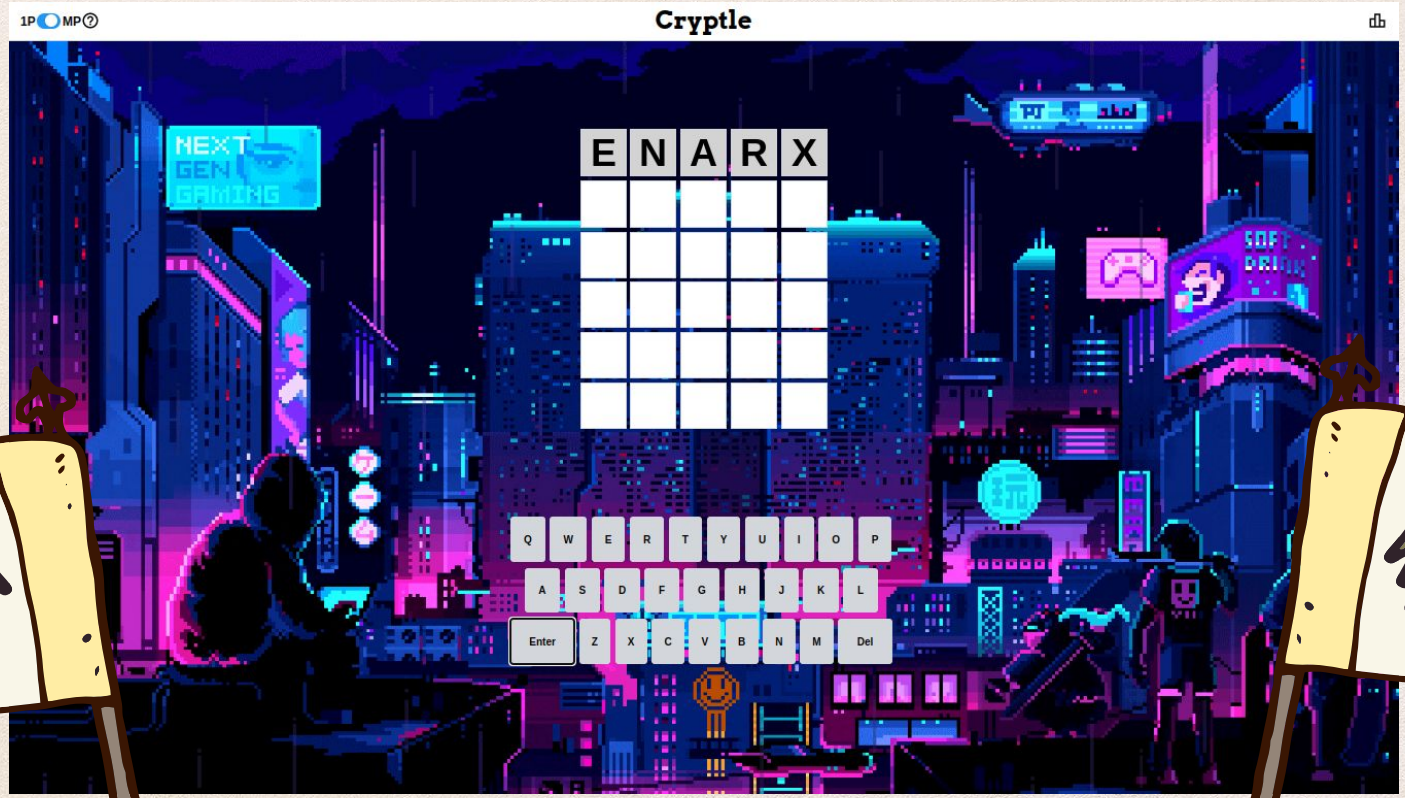
“Confidential Computing protects data in use by performing computation in a hardware-based Trusted Execution Environment. These secure and isolated environments prevent unauthorized access or modification of applications and data while in use, thereby increasing the security assurances for organizations that manage sensitive and regulated data.”



02

Cryptle




1P MP
Cryptle
NEXT GEN GAMING
E N A R X
Q W E R T Y U I O P
A S D F G H J K L
Enter Z X C V B N M Del

Goals

Awareness

- To demonstrate data encryption in use, where the processing of the data is done in a TEE, and only accessible to the app.

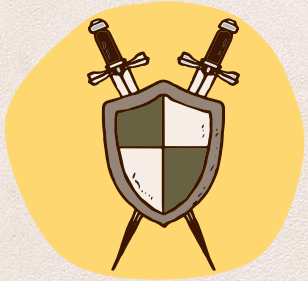
Security

- To help the core team uncover and fix vulnerabilities in the Enarx project, thus increasing the security of the software.

How to play?

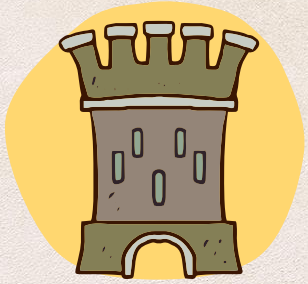
Single Player

Guess 5-letter word
Similar to Wordle



Multi Player

Guess words from others
Words sent to Cryptle
Revealed only when match



Hack Challenge

Guess words from others
With root access on server

Multi Player Demo

Attacking Cryptle on Wasmtime

Failing to Attack Cryptle on Enarx

03

Cryptle Hack Challenge

Participation

Open Source

Has to be open source

Language

May be written in any language

Documentation

Should be provided with explanation

Responsible

Must follow responsible disclosure

Scope

Part of Scope

Enarx runtime
Speculative execution attacks
Timing, side-channel attacks
Breaking out of Wasm sandbox

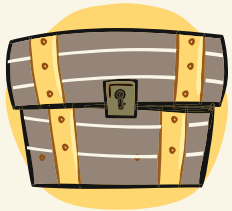
Out of Scope

Cryptle app itself
Hardware/Firmware
Attestation process
Keys for TLS

Process

Run

Run as root for 15 min



Play

Bots submit words

Judged

Judges evaluate

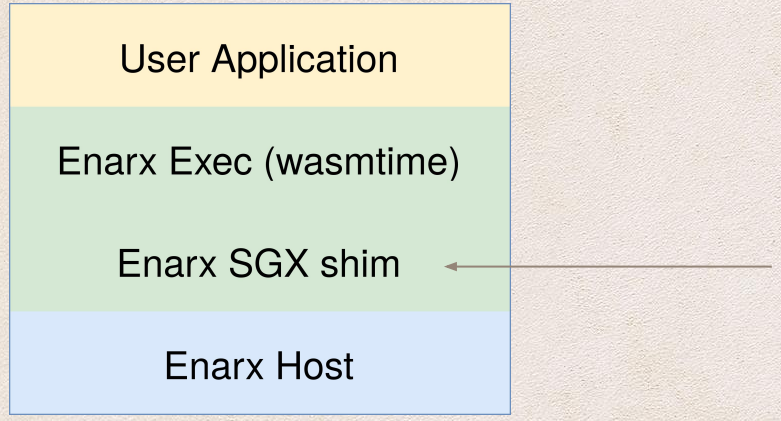
Win

Winners get prizes

04

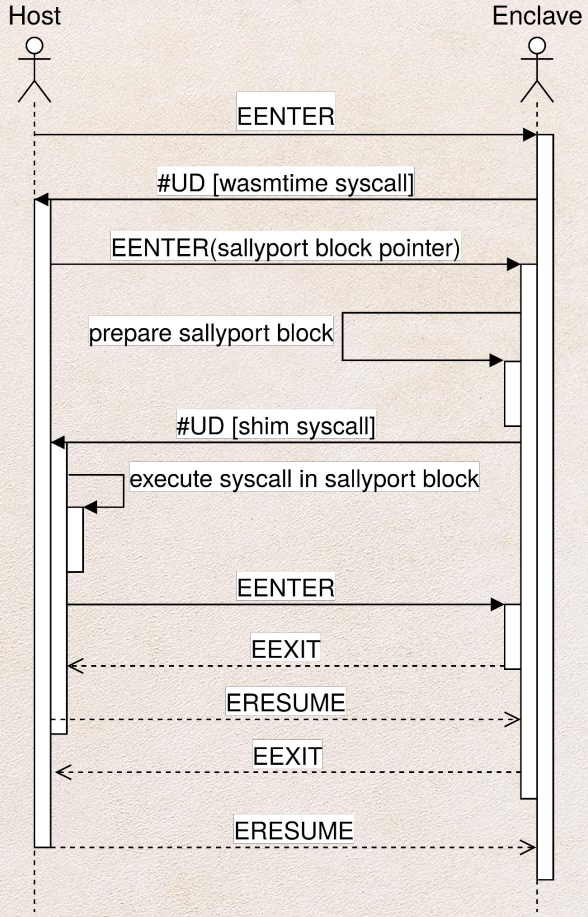
Exploit Walkthrough

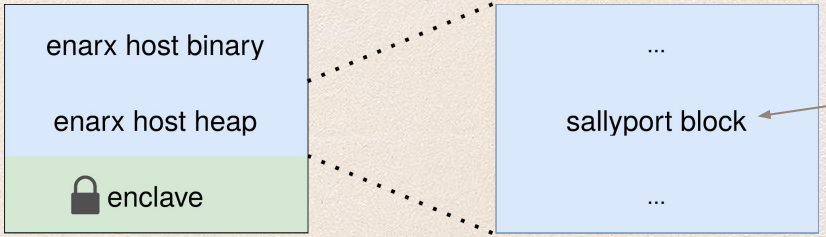
What are we attacking?

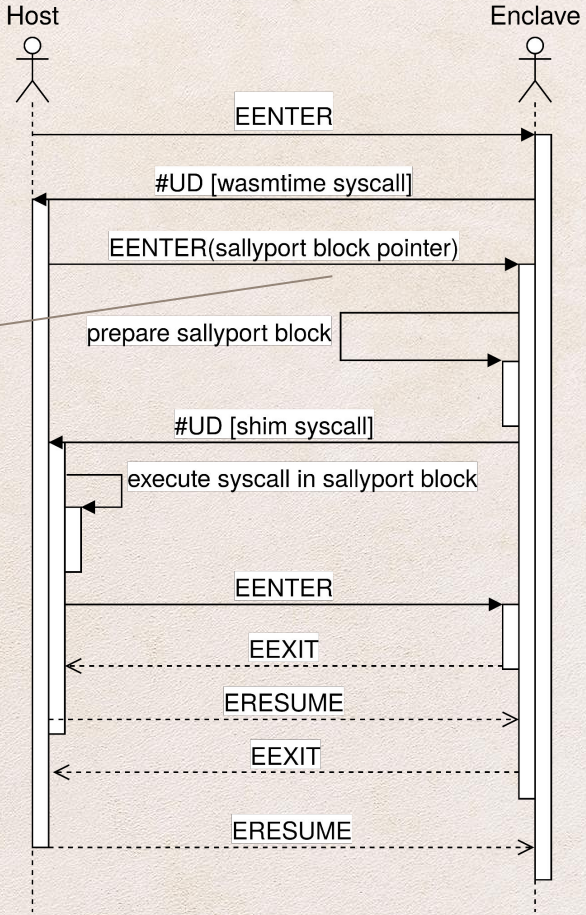


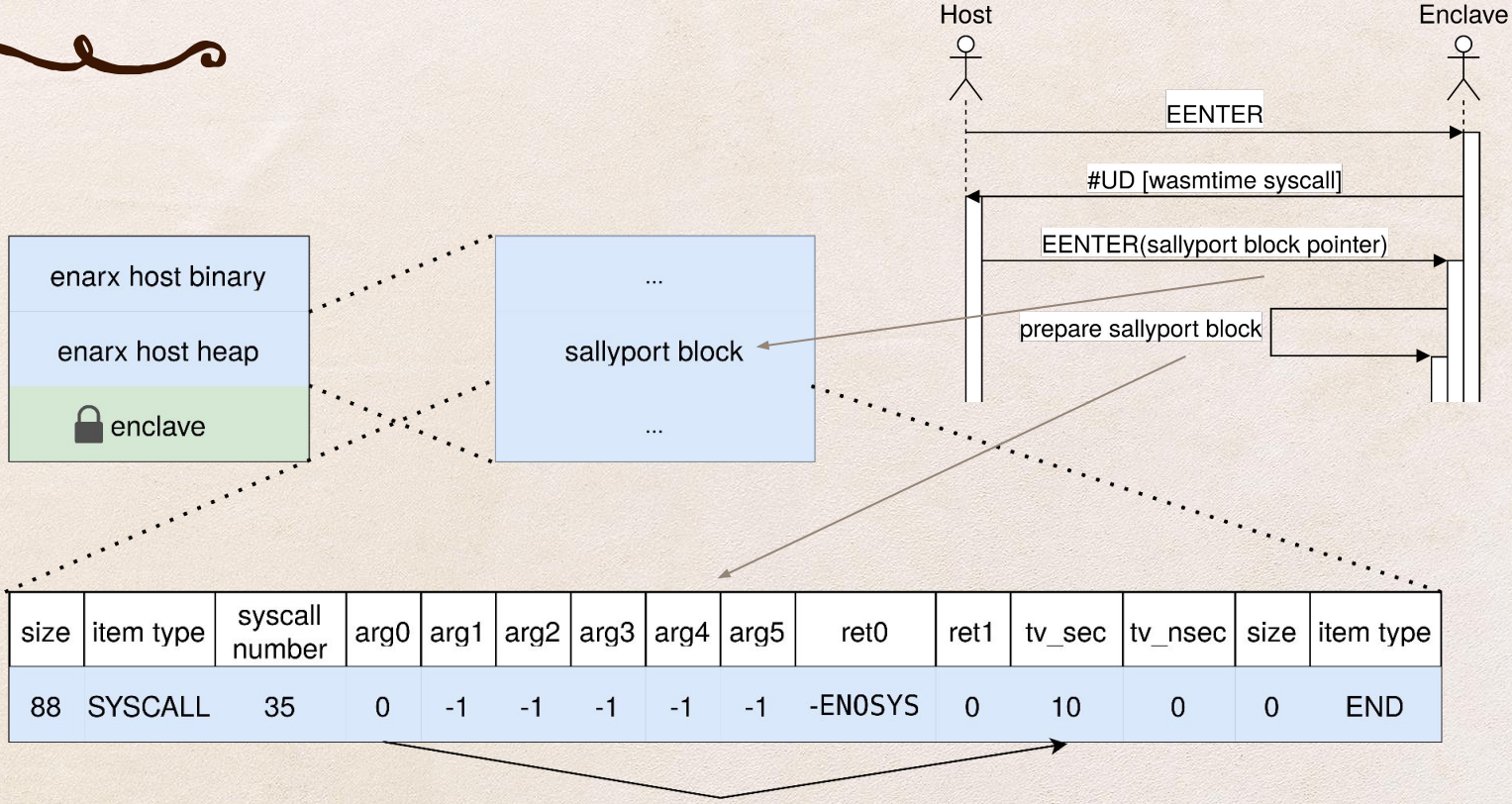
Host-Enclave Communication

a short intro to sallyport on SGX




enarx host binary
enarx host heap
enclave
...
sallyport block
...

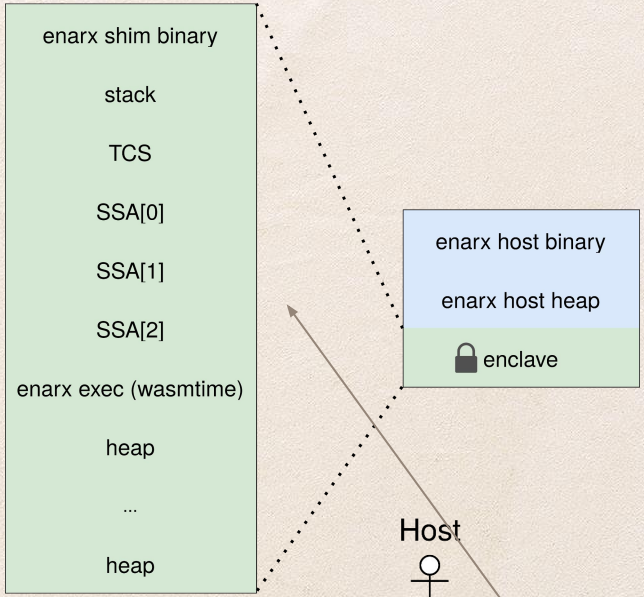

Host
Enclave
EENTER
#UD [wasmtime syscall]
EENTER(sallyport block pointer)
prepare sallyport block
#UD [shim syscall]
execute syscall in sallyport block
EENTER
EEXIT
ERESUME
EEXIT
ERESUME

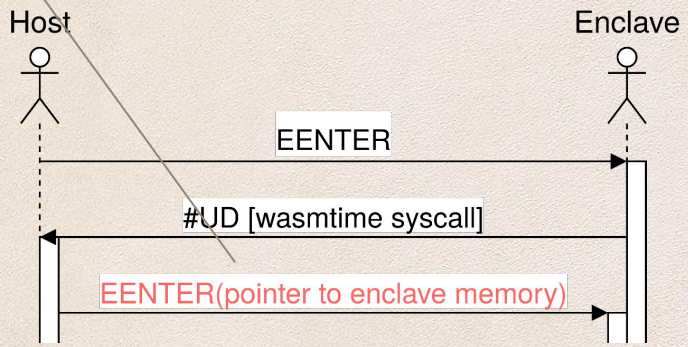


size	item type	syscall number	arg0	arg1	arg2	arg3	arg4	arg5	ret0	ret1	tv_sec	tv_nsec	size	item type
88	SYSCALL	35	0	-1	-1	-1	-1	-1	-ENOSYS	0	10	0	0	END

Vulnerability

The enclave code just assumes that the sallyport block pointer points to host memory.





Vulnerability

The enclave code just assumes that the sallyport block pointer points to host memory.

Conditions

Good for exploitation:

- No ASLR
- We can map arbitrary host memory
- Large binaries with lots of ROP gadgets in memory
- The Enarx Exec can execute regular syscalls
  - ➢ No need to mess around with SGX

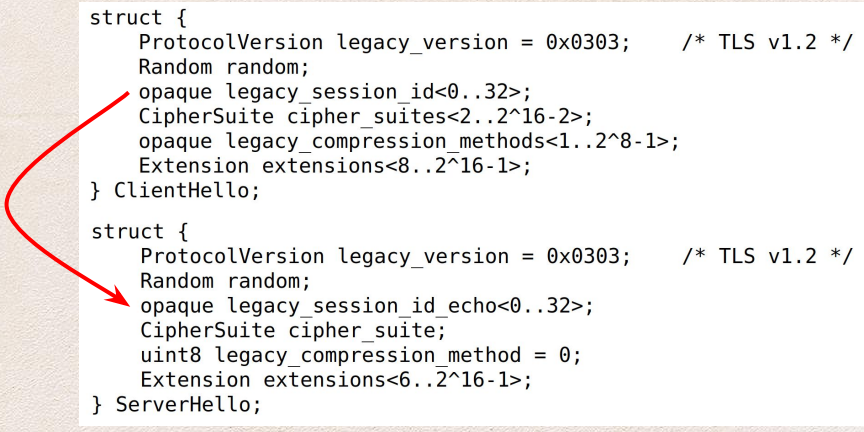
Bad for exploitation:

- Executing instructions located host memory causes a general protection fault
- Enarx enforces W^X
- We can only corrupt memory with the contents of a syscall

Achieving reliable memory corruption

The client's session id is echoed back by the server.

→ We can precisely corrupt up to 32 bytes.

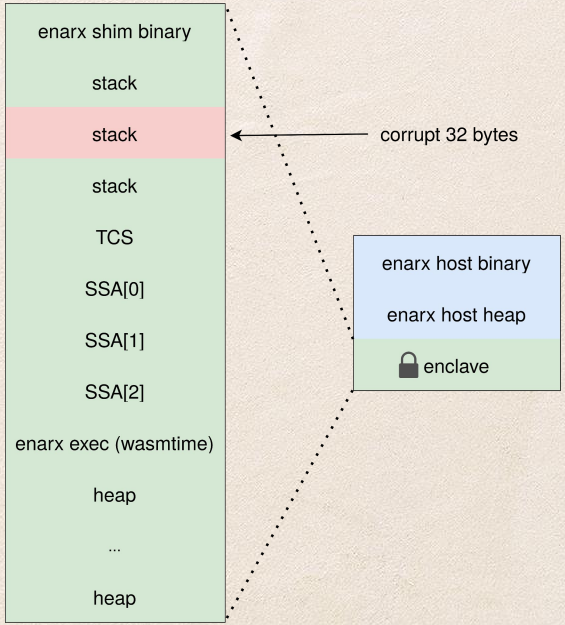


```
struct {
    ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;

struct {
    ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
    Random random;
    opaque legacy_session_id_echo<0..32>;
    CipherSuite cipher_suite;
    uint8 legacy_compression_method = 0;
    Extension extensions<6..2^16-1>;
} ServerHello;
```

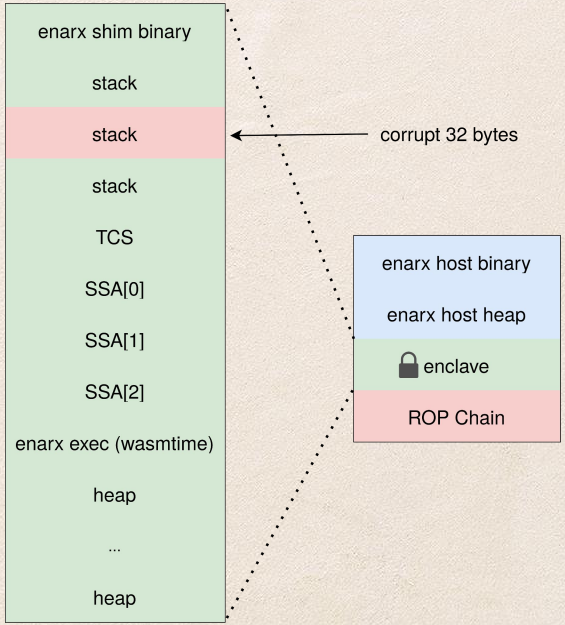
Source: RFC 8446

ROP Easy Mode



- 32 bytes is far too little for a ropchain

ROP Easy Mode



- 32 bytes is far too little for a ropchain
  - The host can map a longer ROP chain into host memory
  - 32 bytes is enough for a ROP chain that switches the stack

Making shellcode executable

The Enarx Exec is a regular linux binary executing syscalls.

The SGX shim intercepts the syscalls and handles them.

→ If we corrupt the stack of the Enarx Exec we use syscalls in our ROP chain and rely on the SGX shim to forward them to the host.

→ mprotect

Final exploit

1. Use vulnerability + session id trick to write a small intermediate ROP chain to the Enarx Exec's stack
2. Switch to a different stack in host memory
3. Write shellcode to enclave memory
4. Execute mprotect syscall to mark the shellcode as executable
5. Jump to the shellcode

Exploit

The Intel SGX shim communicates with the host through a block of host memory called the "sallyport block".

When the host enters the SGX enclave it passes a pointer to the sallyport block in the rdi register.

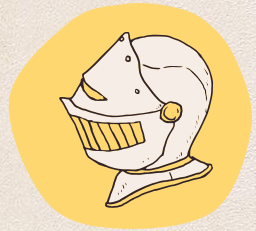
When the shim wants to execute syscalls or other enarx specific commands, it writes the parameters to the sallyport block and passes control back to the host.

The bug is that the shim never checks that the pointer to the sallyport block passed in by the host actually points to host memory and not enclave memory.

Exploit

By passing in a pointer to the enclave's memory the host can trick the shim into corrupting it's own memory.

By starting a TLS handshake with a specially crafted legacy_session_id I can trick the shim into writing up to 32 bytes of our choosing.



In order to manipulate the passed in sallyport block pointer, I use ptrace to read and write to the host registers and memory and intercept host syscalls.

Intel SGX only disallows executing code in non-enclave memory but doesn't disallow executing on a stack in non-enclave memory, so this works perfectly.

Attacking Enarx

https://github.com/Freax13/enarx-exploit

Mitigation

Check that the passed in sallyport block pointer doesn't point to enclave memory.
→ This is easy because we know the size of the enclave.

Thanks!

Please star our project:
github.com/enarx/enarx