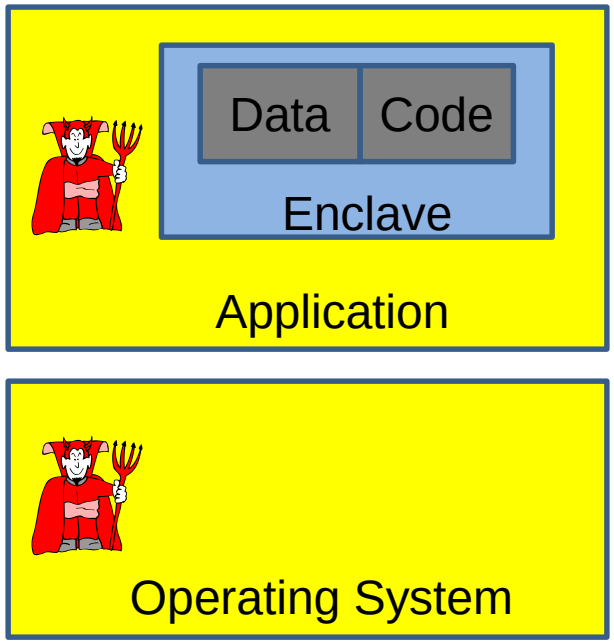
EnGarde: Mutually Trusted Inspection of SGX Enclaves

Hai Nguyen - **Rutgers University**

Vinod Ganapathy - **Rutgers University**

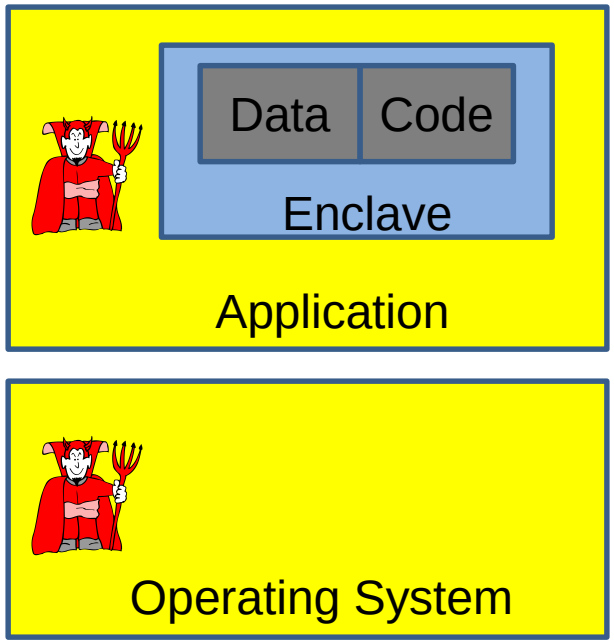
Intel SGX

- Intel Software Guard Extensions (Intel SGX) is an extension of the Intel architecture
- Code and data are kept confidential inside an ***enclave***



Intel SGX

- Protect against privileged software and hardware attacks
- Offer remote attestation to prove secure initialization of the enclave



Problem: Policy compliance

Cloud providers cannot inspect the client's code for policy compliance

- On non-SGX platforms, a benign provider can verify the client's code and data to enforce policy compliance:
  - Code Instrumented with certain security checks to prevent attacks: stack protection, indirect function call checks,…
  - Code is linked against specific versions of certain libraries: The versions of OpenSSL that are free from the HeartBleed exploit,…

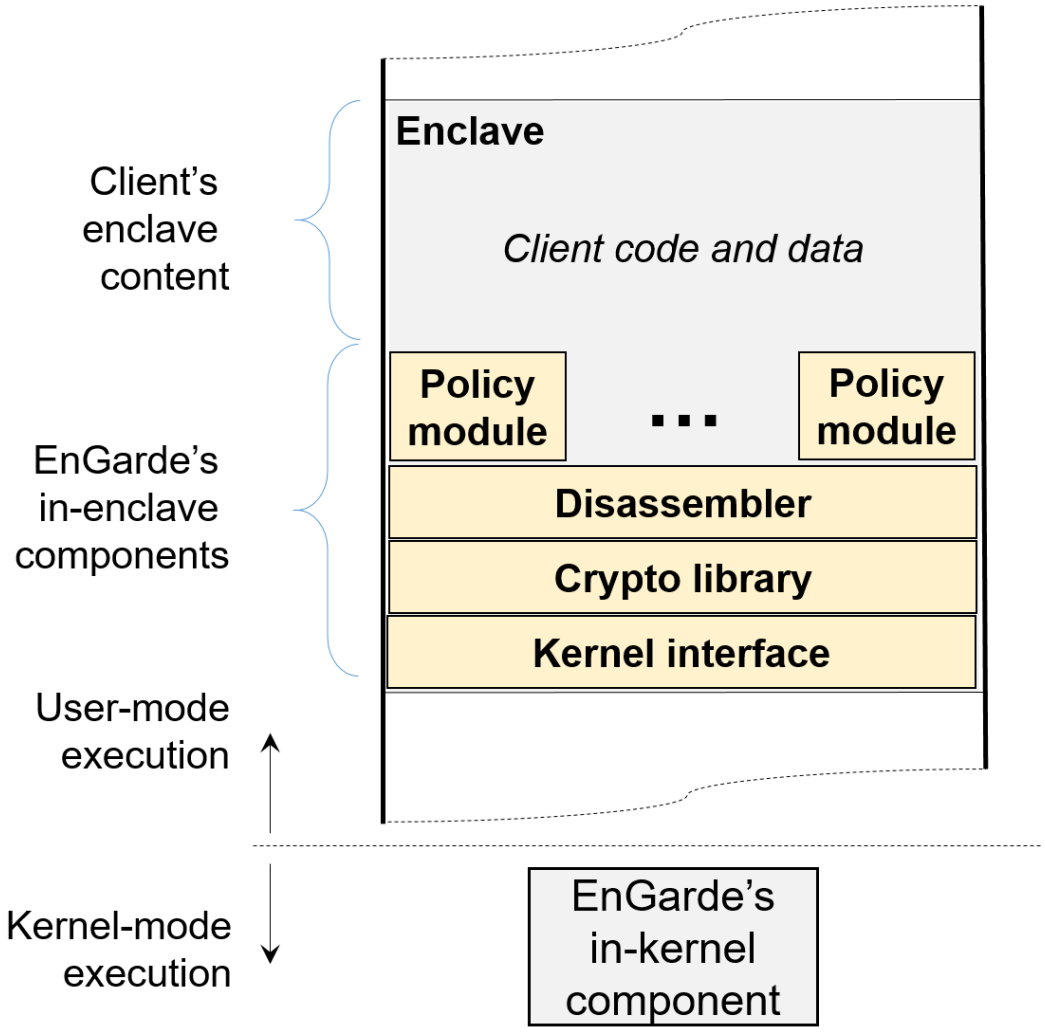
Contributions of our work

- We build EnGarde, an enclave inspection library that allows cloud providers to verify clients' code while preserves the security properties of SGX
- EnGarde incurs small overhead during code provisioning and no overhead during runtime
- We evaluate the effectiveness of EnGarde on various real-world applications

Threat model

- The provider and client are mutually distrusting
- The code of EnGarde is available to both the provider and client for inspection
  - The client verifies that EnGarde does not leak confidential information to the provider
- EnGarde does not consider covert channels via which information about the client's code is leaked to the provider

EnGarde architecture



Enclave initialization and remote attestation

- EnGarde is loaded into a fresh enclave created by the cloud provider

- The provider proves to the client that the enclave was initialized securely by using SGX's remote attestation

Code provisioning

- EnGarde generates a 2048 RSA key pair and sends the public key to the client

- The client uses the public key to encrypt its AES key which will be used to encrypt the sensitive content it sends to EnGarde
  - The content includes code and data represented in an executable using ELF format
  - The executable is compiled as position independent code (PIC) and is statically linked

Code provisioning

- The client sends encrypted content to EnGarde

- EnGarde decrypts the content to get an in-enclave representation of the client's executable

Code disassembly

- EnGarde extracts all code sections of the client's executable and disassembles the code
- EnGarde's disassembler is based on the disassembler of Google's native client (NACL), a sandbox for native code
- EnGarde uses an instruction buffer to store all disassembled instructions

Policy enforcement

- The provider and the client mutually agree upon the policies that the client's code must satisfy

- The agreed policies are encoded into policy modules which are loaded into the enclave along with EnGarde

Policy enforcement

- Policy modules enforce policy compliance by using the disassembled instructions from the instruction buffer
- In general, policy modules examine structural properties of the client's code
- The client's code is rejected if it is not policy compliant

Loading and relocating

- EnGarde maps the *text*, *data* and *bss* segments to the enclave memory

- EnGarde applies symbol relocations using relocation tables

Enclave page permission enforcement

- Page Permission Enforcement is performed by the in-kernel component of EnGarde
- The in-kernel component receives a list of code and data pages which need to be set with appropriate permissions
- Code's pages are set as executable but not writable and the pages of the data segment and bss segment are set as writable but non-executable

Enclave page permission enforcement

- Enclave page permission is enforced at two levels:
  - Using page table permission bits
  - Manipulating the entries of an SGX's data structure called Enclave Page Cache Map (EPCM)

Implementation

- We implemented EnGarde on top of OpenSGX, an SGX emulation infrastructure
- OpenSGX offers rich operating system support and an easy to use library interface for enclave developers
- Current version of Intel SGX does not allow changing enclave page permission at the SGX level after enclave initialization and all enclave memory must be committed at build time

Evaluation

- Goals:
  - Demonstrate the effectiveness of EnGarde and the overhead of EnGarde in enforcing various policies
- Dell Optiplex running Ubuntu 14.04
  - 16 GB RAM
  - Intel core i5 CPU
- Use real world applications: Nginx, Memcached, Netperf, Otp-gen, graph-500, 401.bzip2 and 429.mcf

Sizes of the components of EnGarde

Components	LOC
Code Provisioning	270
Loading and Relocating	188
Checking Executables Linked Against musl-libc	1,949
Checking Executables Compiled With Stack Protection	109
Checking Executables Containing Indirect Function-Call Checks	129
Client's Side Program	349
musl-libc	90,728
Lib Crypto (OpenSSL)	287,985
Lib SSL (OpenSSL)	63,566
Total	453,349

Compliance for library linking

- Verify if applications are linked against musl-libc v1.0.5

- The policy module has the SHA-256 hashes of all the functions of musl-libc v1.0.5 and store them in a hash table

Compliance for library linking

- The policy module uses the instruction buffer and computes the target of each direct function call

- If the hash of a function does not match its value in the hash table, the client has not provided the required musl-libc

Performance of EnGarde to check the library-linking policy

Benchmark	#Inst.	Disassembly	Policy Checking	Loading and Relocation
Nginx	262,228	694,405,019	1,307,411,662	128,696
401.bzip2	24,112	34,071,240	148,922,245	4,239
Graph-500	100,411	140,307,017	246,669,796	4,582
429.mcf	12,903	18,242,127	123,895,553	4,363
Memcached	71,437	137,372,517	489,914,732	8,115
Netperf	51,403	90,616,563	367,356,878	18,090
Otp-gen	28,125	42,823,024	198,587,525	5,388

More results in the paper

- Performance of EnGarde to check the stack protection policy

- Performance of EnGarde to check the indirect function call policy

Conclusion

- EnGarde effectively enforces policy compliance of clients' enclave content
- EnGarde preserves the security properties of SGX
- EnGarde incurs no runtime overhead

Q&A

Backup Slides

Compliance for stack protection

- Compilers emit extra code to protect against stack smashing

- The LLVM compiler adds a guard variable when a function starts and checks the variable when a function exits:

```
19311: mov %fs : 0x28, %rax
1931a: mov %rax, (%rsp)
193fe: mov %fs : 0x28, %rax
19407: cmp(%rsp), %rax
1940b: jne 1941f
1941f: callq 8d5bf <__stack_chk_fail>
```

Compliance for stack protection

- The policy module iterates through the instruction buffer and identifies each function

- Within each function, the policy module checks if stack protection instructions are added at the beginning and at the end of the function

Performance of EnGarde to check the stack protection policy

Benchmark	#Inst.	Disassembly	Policy Checking	Loading and Relocation
Nginx	271,106	719,360,640	713,772,098	128,662
401.bzip2	24,226	34,292,136	862,023,613	4,206
Graph-500	100,488	140,588,361	195,218,892	4,548
429.mcf	12,985	18,288,921	31,459,881	4,330
Memcached	71,677	137,877,497	325,442,403	8,081
Netperf	51,868	91,577,335	183,274,713	18,057
Otp-gen	28,217	43,053,386	217,302,816	5,355

Restricting indirect function calls

- Control flow integrity (CFI) is a measure that guards against attacks that overwrite function pointers to change the flow of a program

- Indirect Function-Call Checks (IFCC) protects indirect function calls by adding code at indirect call sites to transform function pointers to point within a jump table

Restricting indirect function calls

- LLVM implementation of IFCC emits extra code:

```
1b459: lea 0x85c70(%rip), %rax
#<__llvm_#jump_instr_table_0_1>
1b460: sub %eax, %ecx
1b462: and $0x1ff8, %rcx
1b469: add %rax, %rcx
1b475: callq *%rcx
```

- The policy module uses the instruction buffer to look for all indirect function calls and verifies that before each indirect call there is a sequence of instructions *lea*, *sub*, *and* and *add* with relevant data dependence between registers

Performance of EnGarde to check the indirect function-call policy

Benchmark	#Inst.	Disassembly	Policy Checking	Loading and Relocation
Nginx	267,669	821,734,999	20,843,253	128,668
401.bzip2	24,201	34,235,817	1,751,276	4,206
Graph-500	100,424	140,429,738	7,014,913	4,548
429.mcf	12,903	18,242,127	1,177,429	4,330
Memcached	71,508	138,231,446	5,301,168	8,081
Netperf	51,431	91,161,601	3,775,318	18,057
Otp-gen	28,132	42,829,680	2,334,847	5,355