Data flow analysis for Uranus applications

Chan Kwan Yin

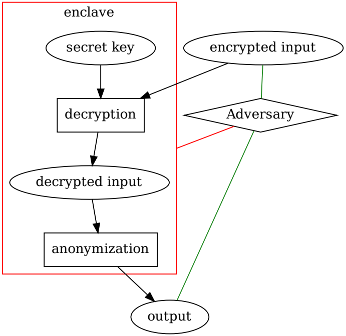
December 14, 2020

Outline

Background

SGX Enclaves

- Servers outsourced to third-party cloud providers
- Threat model: Adversaries with privileged access to OS, BIOS or hardware
- Enclave protects both code and memory from these adversaries



Uranus [3]

- OpenJDK fork that supports Intel SGX
- Methods marked as `@JECall` enters enclaves until return
- Methods marked as `@JOCall` exits enclaves until return
- Useful for integration with libraries like Hadoop and Spark
- Question: Where should `@JECall` and `@JOCall` be placed?
- Question: Is code in these libraries safe as enclave code?

The problem: Performanc/Security Tradeoff

- More code outside enclave:
  - Increased risk of leaking protected data
  - Some leaks may come from unexpected side channels
- More code into enclave:
  - Limited EPC (Enclave Page Cache)
    - Up to 100 MB of EPC
    - Out of memory $\implies$ extremely slow swap
    - JVM applications especially memory-greedy
  - Principle of Least Privilege
    - Vulnerabilities in enclave code bypass enclave protection
    - Vulnerabilities in code outside must only use specific entry points
    - Reduce attack surface

The solution

- *enclavlow* [1]: an information flow analysis tool
- Sources of sensitive data marked with `sourceMarker`
- Anonymization marked with `sinkMarker`
- Identity functions; expected to be optimized them away by JIT

```
@JECall
static int process(byte[] encrypted) {
  byte sum = 0;
  byte[] password = sourceMarker(new byte[]{1, 2, 3, 4, 5, 6});
  byte[] decrypted = decrypt(password, encrypted);
  for(byte b : decrypted) sum ^= b;
  return sinkMarker(sum);
}
```

[1] coined from the words "enclave" and "flow"

Approach

Intuition: Trivial ways of leaking data

- Returning/throwing out of a `@JECall`
- Passing into a `@JOCall`
- Assigning to a static field

Intuition: Non-trivial ways of leaking data

- Assigning to an outside-enclave object:
```
@JECall void x(Manager m) {
  m.value = sourceMarker(secret); }
```
- Leaking control flow into variables:
```
for(int i=0; i < sourceMarker(secret); i++) outside++;
```
- Implicit exceptions:
```
int[3] array; array[sourceMarker(secret)] = 1;
```

Flow analysis

- Analysis framework: Soot [2]
- Each method is analyzed independently
    - Function calls treated as blackboxes
    -
-

Flow graph

(missing graphic) (missing graphic)

Limitations & Future Work

Detecting implicit exceptions

- Conditional runtime errors leaks data
- Extensively researched in both academia and industry
- Solutions usually achieved at the language level, e.g. `@NonNull`, `@Size`
- Good practices: all exceptions should be caught at enclave boundary anyway!
- Similar: `x + secret - secret`

Tackling polymorphism

- Computing all combinations of instance classes takes exponential time.
- *"Java workloads don't fit into enclave programming paradigms"* [1]
- Detecting possible paths subclasses reduces complexity, but still not perfect.

Integration into Uranus

- Uranus disallows reads/writes of objects outside enclave without `SafeGetField` etc
- Runtime overhead of checking object location
- *enclavlow* to perform this analysis at compile-time

Conclusion

- Assist with decisions on performance/security tradeoff
- Incorporated into Uranus
- Applications in big data industry
- Room for improvement on specialization cases

Appendix

References

CHE TSAI, C., SON, J., JAIN, B., MCAVEY, J., POPA, R. A., AND PORTER, D. E.
Civet: An efficient java partitioning framework for hardware enclaves.
In *29th USENIX Security Symposium (USENIX Security 20)* (Aug. 2020), USENIX Association, pp. 505–522.

EINARSSON, A., AND NIELSEN, J. D.
A survivor's guide to java program analysis with soot.
BRICS, Department of Computer Science, University of Aarhus, Denmark 17 (2008).

JIANG, X. J., TZS, C., LI, O., SHEN, T., AND ZHAO, S.
Uranus: Simple, efficient sgx programming and its applications [unpublished].
In *Proceedings of the 15th ACM ASIA Conference on Computer and Communications Security (ASIACCS '20)* (2020).