Towards Memory Safe Python Enclave for Security Sensitive Computation

Huibo Wang, Mingshen Sun, Qian Feng, Pei Wang, Tongxin Li, Yu Ding

Abstract

Intel SGX Guard eXtensions (SGX), a hardware supported trusted execution environment (TEE) is designed to protect security sensitive applications. However, since enclave applications are developed with memory unsafe languages such as C/C++, the traditional memory corruption is not eliminated in SGX. Rust-SGX is the first toolkit providing enclave developers with a memory safe language. However, Rust is considered a Systems language and has become a good choice for concurrent applications and web browsers. Many application domains such as Big Data, Machine Learning, Robotics, Computer Vision are more commonly developed in Python programming language. Therefore, Python application developers cannot benefit from secure enclaves like Intel SGX and Rust-SGX.

To fill this gap, we propose Python-SGX which is a memory safe SGX SDK providing enclave developers a memory safe Python development environment. The key idea is to enable a memory safe Python language in SGX by solving the following key challenges: (1) defining a memory safe Python interpreter (2) replacing unsafe elements of Python interpreter with safe ones, (3) achieving comparable performance to non-enclave Python applications, and (4) not introducing any unsafe new code or libraries into SGX. We propose to build Python-SGX with PyPy, a Python interpreter written by RPython which is a subset of Python, and tame unsafe parts in PyPy by formal verification, security hardening, and memory safe language. We have implemented Python-SGX and tested with a series of benchmarks programs. Our evaluation results show that Python-SGX does not cause significant overhead.

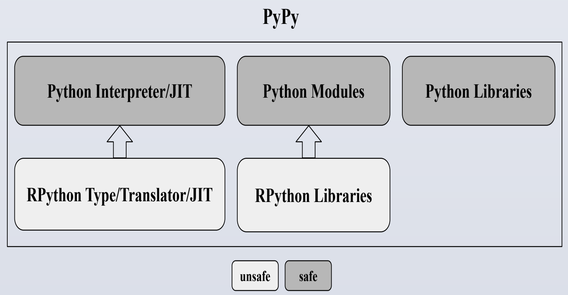
Objectives

- A secure SGX enclave environment for Python software development free of memory corruptions
- Capabilities and rich library support as in native Linux environment
- No significant performance overhead

Challenges

- Memory safety of Python interpreter
  - CPython has 300k lines of C code
  - PyPy, a Python interpreter is written in RPython which is a subset of Python, has three unsafe parts related to RPython
- Build from scratch or on top of Intel SGX SDK
- Porting PyPy interpreter in Intel SGX without introducing extra memory unsafe parts

Overview of PyPy



Threat Model and Scope

Only the software running inside the enclave is trusted, and the rest is not trusted

✓ No extra mechanisms to defeat side channel attacks
✓ Focusing on enable application layer memory safety, lower layer software are not within the scope

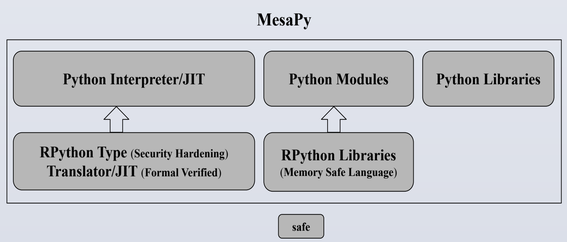
Key Insight

Unsafe PyPy

- Spatial safety risk in RPython type system
- C code in RPython's Translator/JIT backend
- RPython external C libraries

Memory Safe PyPy (MesaPy)

- Security Hardening (list and array index)
- Formal Verification tools (buffer overflow, buffer over-read, null pointer dereference, memory leak)
- Memory Safe Language



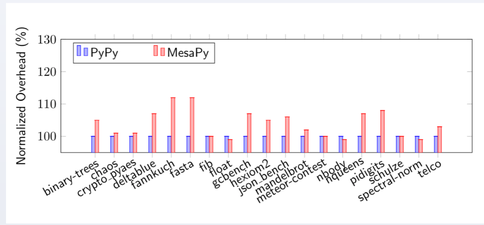
Porting MesaPy to Intel SGX

- I/O
- System calls
- ...

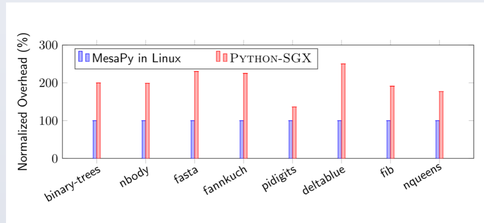
- Environment variables, Disk files, Dynamic loading
- Previous works have focused on building bridge for the limitation, however it increases the TCB and also adds an extra layer inside SGX which is written in C/C++.

Evaluation

MesaPy normalized against PyPy native execution



Python-SGX normalized against MesaPy native execution



References

1. C.-C. Tsai, D. E. Porter, and M. Vij, “Graphene-sgx: A practical library os for unmodified applications on sgx,” in Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference , ser. USENIX ATC ’17. Berkeley, CA, USA
2. H. Wang, E. Bauman, V. Karande, Z. Lin, Y. Cheng, and Y. Zhang, “Running language interpreters inside sgx: A lightweight,legacy-compatible script code hardening approach,” in Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security , ser. Asia CCS ’19. New Zealand.
3. H. Wang, P. Wang, Y. Ding, M. Sun, Y. Jing, R. Duan, L. Li, Y. Zhang, T. Wei, and Z. Lin (2019). Towards memory safe enclave programming with rust-sgx. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS ’19, London, UK.
4. A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas, “The seahorn verification framework,” 2015.
5. Z. Rakamarić and M. Emmi, “SMACK: Decoupling source language details from verifier implementations,” in Proceedings of the 26th International Conference on Computer Aided Verification (CAV), ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, 2014, pp. 106–113.