Securing OS Binaries:
TEE-Based Progressive Randomization

Armand-Alexandru Balint, Arto Niemi

What is code randomization?

Diversification method that alters a program's executable memory layout to make run-time addresses unpredictable.

What is fine-grained randomization?

Unlike traditional ASLR which shuffles large memory segments (code, stack, heap) and where a single leak often de-randomizes an ASLR segment; fine-grained preserves internal unpredictability. It reorders smaller, well-defined code units within these segments. This creates an unpredictable internal layout, sizably increasing attacker effort even if segment base addresses are known.

Why is it necessary?

Roughly **70%** of security flaws come from memory safety, a problem theoretically solvable, but practically insurmountable across existing, extensive compiled software. While operating systems like Linux, MacOS, and Windows randomize base addresses, it is not enough against these flaws, making fine-grained randomization necessary for legacy codebase.

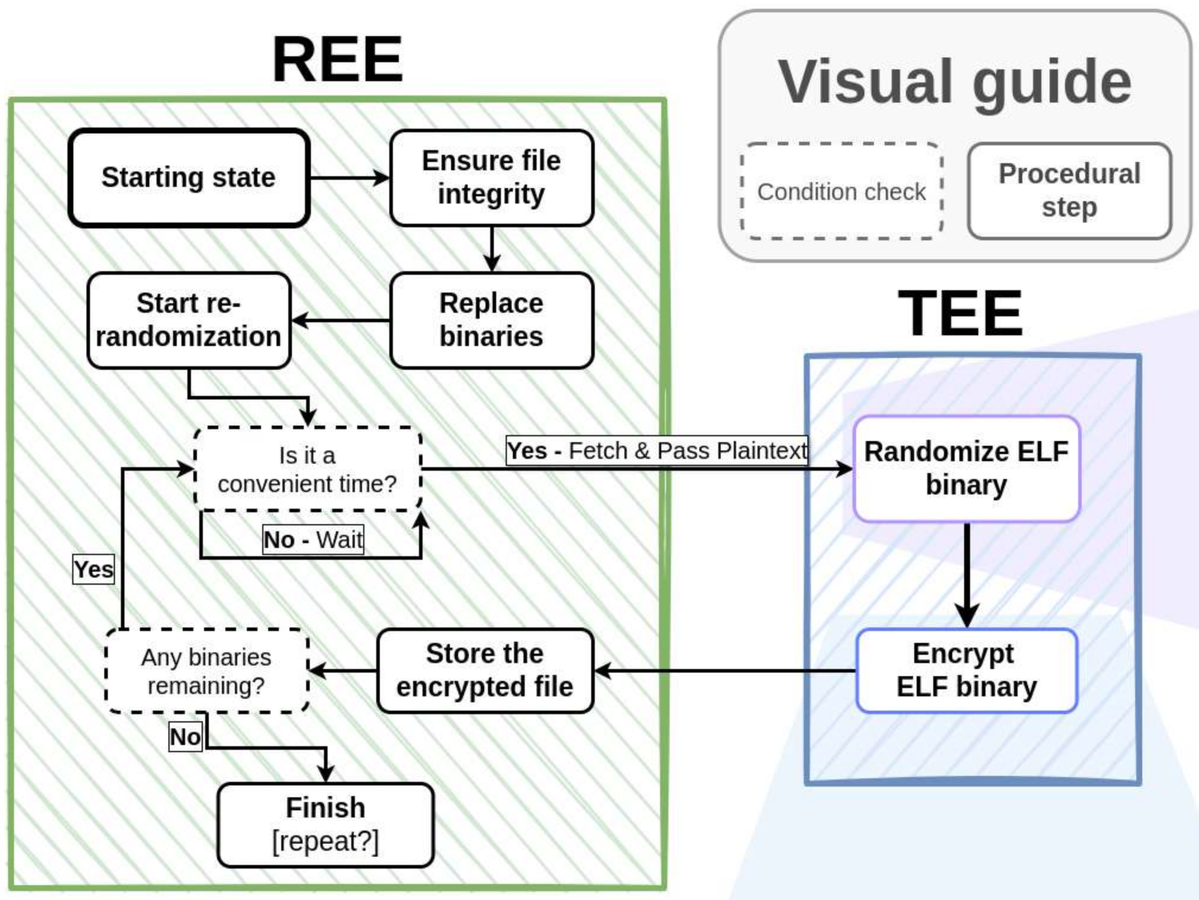
Design criteria:

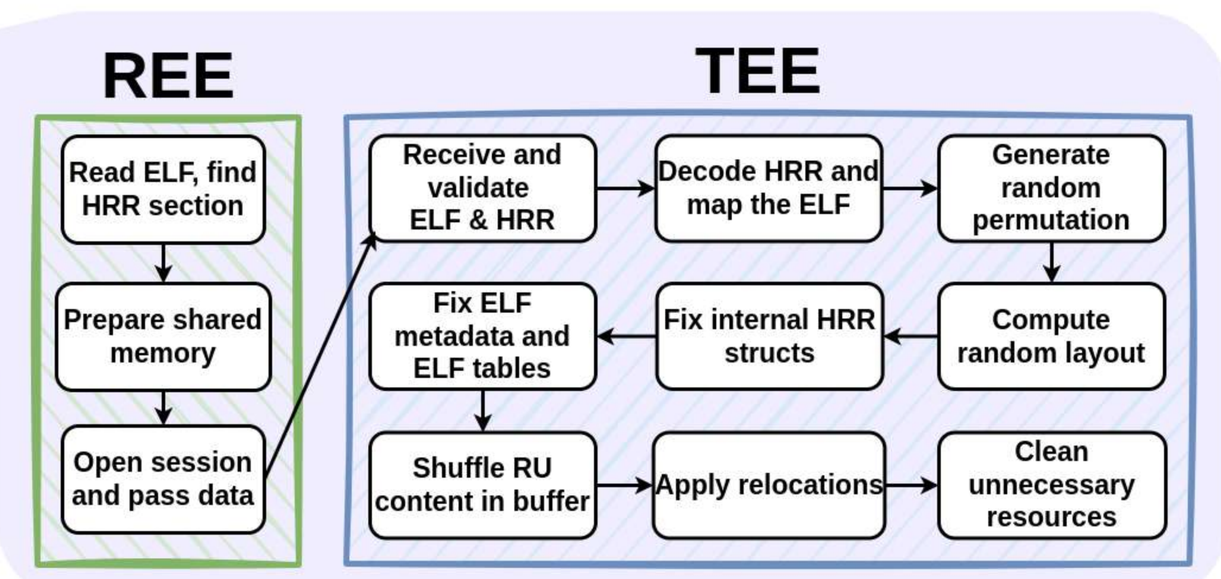
Efficiency: Additional overhead must be negligible
Secrecy: Sensitive data must never enter user space
Integrity: Interruptions of any must not corrupt any data
Uniqueness: Nonces must be unique and never be reused

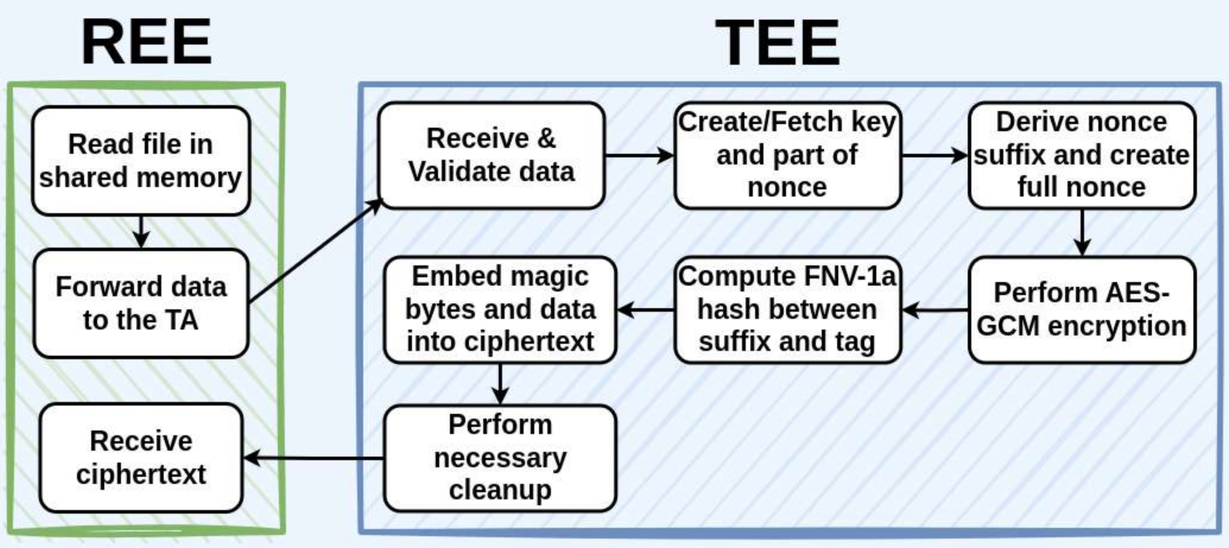
Assumed attacker capabilities:

I) Persistent access to plaintext files on disk
II) Code-reuse attack against a live process
III) Information disclosure attack against a live process

High-level approach



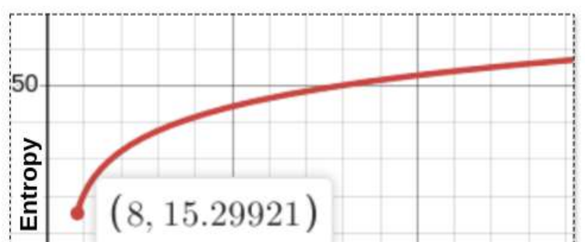




Entropy

Our randomization process operates on distinct code segments termed Randomization Units (RUs). By permuting these RUs, we achieve a combinatorial complexity of |RU|!, leading to an entropy of $\log_2$ (RU!) bits from the shuffling scheme itself. However, when an attack requires a chain of k specific gadgets, each residing in different RUs, the complexity of locating all k RUs in their new, permuted positions can be approximated by an entropy of $\log_2\left(\frac{N!}{(N-k)!}\right)$ where N denotes the RUs.

Assuming an attacker would need 8 gadgets for an attack, entropy growth based on the number of RUs in the binary is depicted on the right



End goal

Prevent passive memory attacks and maximally reduce active exploit surfaces with negligible performance overhead. We achieve this by protecting ELF binaries through a TEE-interfacing preloader for secure, metadata-driven decryption and randomization.

Helsinki System Security Lab (HSSL)

HSSL drives renewal and mastery in the field of platform and device related security technologies, especially for Huawei consumer devices such as mobile phones, laptops, televisions and automotive. We do research in topics such as hardware-assisted isolation and integrity, as well as in operating system protection (hypervisor, TEE, secure enclaves and kernel hardening). We also carry expertise in cryptography and systems security functionality such as device key management (PKI), device attestation and key-store solutions.

HUAWEI