ABSTRACT

Cryptography is essential for computer and network security. When cryptosystems are deployed in computing or communication systems, it is extremely critical to protect the cryptographic keys. In practice, keys are loaded into the memory as plaintext during cryptographic computations. Therefore, the keys are subject to memory disclosure attacks that read unauthorized data from RAM. Such attacks could be performed through software exploitations, such as OpenSSL Heartbleed, even when the integrity of the victim system's binaries is maintained. They could also be done through physical methods, such as cold-boot attacks, even if the system is free of software vulnerabilities. This paper presents Mimosa, to protect RSA private keys against both software-based and physical memory disclosure attacks. Mimosa uses hardware transactional memory (HTM) to ensure that (a) whenever a malicious thread other than Mimosa attempts to read the plaintext private key, the transaction aborts and all sensitive data are automatically cleared with hardware, due to the strong atomicity guarantee of HTM; and (b) all sensitive data, including private keys and intermediate states, appear as plaintext only within CPU-bound caches, and are never loaded to RAM chips. To the best of our knowledge, Mimosa is the first solution to use transactional memory to protect sensitive data against memory attacks. However, the fragility of TSX transactions introduces extra cache-clogging denial-of-service (DoS) threats, and attackers could sharply degrade the performance by concurrent memory-intensive tasks. To mitigate the DoS threats, we further partition an RSA private-key computation into multiple transactional parts by analyzing the distribution of aborts, while (sensitive) intermediate results are still protected across transactional parts. Through extensive experiments, we show that Mimosa effectively protects cryptographic keys against attacks that attempt to read sensitive data in memory, and introduces only a small performance overhead, even with concurrent cache-clogging workloads.