License Locking

Andrei Coman
Yunzhou Li
Jacob Polatty

Estimated Total Hours: 40

Plans (Past Week)

- Test CryptoFP in different language implementation
- Develop cryptographic algorithm for generating license key from a fingerprint
- Create basic structure for encrypting a binary with the key derived from the fingerprint and decrypting it on the user's machine
- Consider how to securely store the fingerprint on the user's machine so that it cannot be stolen to decrypt the binary

What Actually Happened

Work completed: Investigated implementation of clock_gettime and available clocksources in the Linux Kernel. Noted prevalence of TSC and unavailability of RTC/HPET without sudo. Developed Python implementation of CryptoFP and alternate C++ version using HPET timer to alleviate collisions.

Major challenges+roadblocks:

- All (ns-resolution) userspace functions are TSC based.
- ns-resolution RTC/HPET require higher privileges and destabilize TSC.
- HPET runs much slower than TSC

Attribution:

- Andrei: Kernel digging
- Yunzhou: Testing TSC and HPET on VM
- Jacob: CryptoFP implementation in Python, HPET-based implementation, extensive testing on both GCP and AWS EC2

Clocksource Tradeoffs

- Using the TSC (or a timer derived from it) is stable but CryptoFP is unable to differentiate between machines with the same hardware since these clocks measure the CPU cycle count
  - The default clocksources for GCP and EC2 (kvm-clock/xen) are paravirtualized by the hypervisor but are computed as a multiple + offset from the TSC
- Using the HPET completely eliminates collisions (0% rate) between machines with the same hardware, though it is less stable than the TSC (some fingerprints checks give false negatives)
  - EC2 VMs expose the HPET, which can be manually selected for use during the fingerprinting process (requires a sudo command)
  - The rate of false negatives can be reduced to <10% by forcing CryptoFP to always run on the same CPU core, and this rate does not appear to change when the system is under load
  - One possible mitigation tactic is to take several fingerprint readings and to return a match if one of the samples matches the stored value, this showed very promising results on EC2 but does sacrifice runtime

```
ubuntu@ip-172-31-47-215:~/LicenseLockingV3/cryptofp-cpp$ ./taskset_test.sh --fp=fingerprints/aws_fp_b2 --test_count=100 --stress=1
cpu 0: 1/100 failed
ubuntu@ip-172-31-47-215:~/LicenseLockingV3/cryptofp-cpp$ ./taskset_test.sh --fp=fingerprints/aws_fp_b1 --test_count=100 --stress=1
cpu 0: 100/100 failed
ubuntu@ip-172-31-47-215:~/LicenseLockingV3/cryptofp-cpp$
```

Plans (Next Week)

- Develop cryptographic algorithm for generating license key from a fingerprint
- Create basic structure for encrypting a binary with the key derived from the fingerprint and decrypting it on the user’s machine
- Consider how to securely store the fingerprint on the user’s machine so that it cannot be stolen to decrypt the binary

Summary/Overall Progress

Implementation of CryptoFP achieves

a) Stability on a single machine (for at least one week)
b) Stability across different power settings **AND CPU loads**
c) Fingerprint differentiation between machines with different HW

Competitive implementations are all based on **one single clocksource**.

Improvements:

a) Collisions among different machines with same HW