Writing Constant-Time Code
What, why and how?

Chris Jones (slides from MPG)
2020/11/03
Side-Channel Attacks

• Constant-time code is a countermeasure

• “Attack based on information gained (‘leaked’) from the practical implementation of a system”
  - Can completely break real world encryption

• Many types exist
  - Timing
  - Cache
  - Branch prediction
  - Many more (power, fault, etc...)

© 2020 Arm Limited (or its affiliates)
A Simple Example

• Checking that a MAC (for example) matches an expected value:

```c
int compare(uint8_t *a, uint8_t *b, size_t len)
{
    for (size_t i = 0; i < len; i++)
        if (a[i] != b[i])
            return -1;
    return 0;
}
```

• This is vulnerable to a number of side-channel attacks
A Simple Example

• Checking that a MAC (for example) matches an expected value:

```c
int compare(uint8_t *a, uint8_t *b, size_t len)
{
    for (size_t i = 0; i < len; i++)
        if (a[i] != b[i])
            return -1;
    return 0;
}
```

• This is vulnerable to a number of side-channel attacks
What Can Be Done?

• Use constant-time code to eliminate this class of vulnerability

• Execution time cannot depend on secret values

• Follow the golden rules:
  - No branches depending on secret data
  - No memory access depending on secret data
  - No variable-time instruction executed on secret data

• Limits: physical side-channels, faults, readability
Fixing Our Example

```c
int compare(uint8_t *a, uint8_t *b, size_t len)
{
    for (size_t i = 0; i < len; i++)
        if (a[i] != b[i])
            return -1;
    return 0;
}
```
Fixing Our Example – Removing Branches

```c
int compare(uint8_t *a, uint8_t *b, size_t len)
{
    for (size_t i = 0; i < len; i++)
        if (a[i] != b[i])
            return -1;
    return 0;
}
```
int compare(uint8_t *a, uint8_t *b, size_t len)
{
    char diff = 0;
    for (size_t i = 0; i < len; i++)
        diff |= a[i] ^ b[i];
    return diff;
}
Example #2 – Table Lookup

• Seemingly a constant time instruction - O(1)

```c
u32 lookup(u32 *t, u32 l, u32 secret_i) {
    return t[secret_i];
}
```

• Vulnerable to a cache attack!

• Violates rule #2 (No memory access depending on secret data)
Example #2 – Table Lookup

• Seemingly a constant time instruction - \( O(1) \)

```c
u32 lookup(u32 *t, u32 l, u32 secret_i) {
    u32 r = t[0];
    for (u32 j = 1; j < l; j++) {
        r = choose(r, t[j], eq(secret_i, j));
    }
    return r;
}
```

• Vulnerable to a cache attack!

• Violates rule #2 (No memory access depending on secret data)
Real World Examples


• (2011) “Remote Timing Attacks Are Still Practical”

• (2013) Lucky 13

• (2018) Spectre & Meltdown
Conclusion

• Constant-time code can be used to defend against a variety of side-channel attacks

• Constant-time code is not easy to write - prone to both human and compiler error

• Promising techniques to automate testing & writing
  - Still needs some improvements & widespread adoption
Thank You
Danke
Merci
Merci
Merci
Merci
감사합니다
धन्यवाद
شكرا
ধন্যবাদ
תודה