SWATT

A software-based attestation method for embedded devices
Lecture outline

• Introduction
• Problem definition, assumptions & model
• SWATT
• Future work
Introduction

• Attestation is the ability to affirm to be correct, true, or genuine
• We would like to provide the ability to verify the memory content of a device we are about to interact with
• This is one way to establish the absence of malware (viruses worms, trojan horses...
Introduction

• Embedded devices cannot be physically secured & may often be in a hostile surrounding

• Cost is a major issue, even a small increase in device cost leads to a significant increase in high volume production

• Hardware solutions may be expensive

• Devices will typically have no virtual memory. (Kennel & Jamieson method)
Introduction

• Software based attestation method (can be used on legacy devices, no need for special hardware)
• Attests the device code, static data & configuration settings
• The verifier does not require direct (hardware) access to device memory
1. Generate Random challenge
   Precompute result

2. Challenge (+ Attestation Routine)

3. Execute Attestation Routine

4. Challenge Result

5. Compare results
   Measure response time

Introduction

SWATT: A challenge response protocol
Lecture outline

• Introduction
• Problem definition, assumptions & model
• SWATT
• Future work
Definitions

Naïve attempt:

- Verifier sends a challenge, device uses challenge as MAC key & computes MAC on memory, result is returned to verifier

- ✔ Replay attack
- ✔ Pre-computation
- ✗ Malware can cheat – empty memory content can be used to store malware & skipped during the MAC computation. Malware code could also move original code to another device which will be used for MAC computation
Definitions

• MAC is not good enough, can we find an effective verification procedure?
• If the memory contents of the device is the same as the value expected by the verifier the checksum will **always** be correct & verification will be successful
• Verification will fail with high probability if the memory contents of the device differs from the expected content
Assumptions

• Verification procedure exists on the device (or downloaded prior to verification)

• Verifier knows the expected memory layout

• Verifier is intimately familiar with device HW
  – Clock speed
  – Memory architecture
  – Memory size
  – Instruction set (ISA)
Threat model

• Attacker has full control over the device memory

• Attacker does not modify the device HW
  – Increase memory size
  – Change memory access timing
  – Increase CPU clock speed

• Attacker does not tamper with bios
Lecture outline

- Introduction
- Problem definition, assumptions & model
- SWATT
- Future work
SWATT

General concept:
• The random challenge is used to seed the pseudorandom number generator
• Number are used to traverse memory & update MAC
• Malware cannot predict which memory will be requested next (altered memory or original memory)
• An extra if statement must be added to divert load operations from the altered memory
• The increase in running time will be noticeable
Desired verification process properties

• The PRG should be optimized for CPU architecture (rc4 for 8bit, Helix for 32 bit)

• Resistance to replay & pre-computation attacks is achieved by seeding the PRG with a random challenge

• High probability of detecting changes
  – Every memory location must be accessed
    Coupon collector problem shows that we need to make $O(n \ln(n))$, where $n$ is the memory size
  – MAC should reflect changes to a single byte
Desired properties – Cont.

• Small code size – an ‘if’ statement translates to a compare instruction + a conditional branch i.e. ~2-3 CPU cycles. If this is to be noticeable the verification procedure should take few tens of CPU cycles (not inc. init & epilogue)

• Optimized implementation – if attacker can find a more optimized implementation of the verification procedure it will be possible to conceal the extra ‘if’
Desired properties – Cont.

• Non parallelizable – To prevent multiple devices from performing distributed computation, to solve this the actual memory access is based on the RNG & the current checksum value
SWATT

Experimental results

• Genuine software
• Attacker’s version
  – Single byte of modified code
  – Single ‘if’ statement in the verification procedure

![Graph showing experimental results for SWATT]
SWATT

Considerations for practical use:

• Number of iterations

• Architecture:
  – Harvard: only program memory (code + static) needs to be verified. Different read latencies can be serve an attacker
  – Von – Neumann: code & data share memory. How do we handle the data section (stack, sensor readings...)
    • Software must be designed to have checkpoints where data state is predictable
    • Verifier can download data section

• Empty memory regions:
  – should be filled with a random pattern (so that an attacker cannot suppress the read operation & save time)
Future work

• Checksum / RNG
  – Will vary between platforms

• Code Optimization
  – Theoretical framework to proof maximum optimization

• How to perform device attestation remotely
  – Untrusted network
  – Unpredictable networking latencies

• Devices with sophisticated architecture
  – Virtual Memory
  – Branch prediction
SWATTPro

• Try to prevent static analysis to the attestation functions
  – Randomization
  – Encryption
  – Self-modifying code (this is what virus do)
  – Opaque prediction
  – Junk instruction
• Mechanisms 1, 2, 4, and 5 are easy to understand
• Let us look at number 3
  – Three operations: jump, read, and self-modifying
  – The self-modifying segment will determine where to jump and what to read