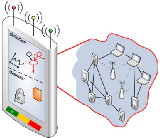
Code Attestation with Compressed Instruction Code

Benjamin Vetter (benjamin.vetter@haw-hamburg.de)
Dirk Westhoff (westhoff@informatik.haw-hamburg.de)

University of Applied Sciences HAW Hamburg
Depts of Information & Electrical Engineering & Computer Science

Stand: 14. Juni 2011

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences



Outline

1 Introduction and motiviation

2 Code attestation protocols

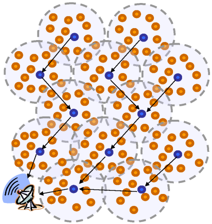
3 Our Approach

4 Risks, Outlook

5 Summary

6 References

Wireless sensor nodes


[1]

[2]

- Applications: monitoring, detection, control
- High flexibility, low cost, low power and security
- Embedded, wireless security research

[1] http://eccwsn.blogspot.com/
[2] http://www.informatik.uni-augsburg.de/

Wireless sensor nodes

Conflict / motivation

Low security VS need for trustworthy sensor data

Desirable:

- 'Secure' wireless sensor nodes
- Apply sensor nodes in more critical areas (e.g. health)


[a]

[a] `http://www.el-stift.de`

Idea: code attestation

Problem:

- Nodes 'easy' to compromise
- Low budget sensor nodes

$\Longrightarrow$ costly trusted platform modules (TPMs) or tamper proof units not applicable

Idea: code attestation

Nodes have to proof their trustworthiness to a trustworthy instance (base station)

Outline

1 Introduction and motiviation

2 Code attestation protocols

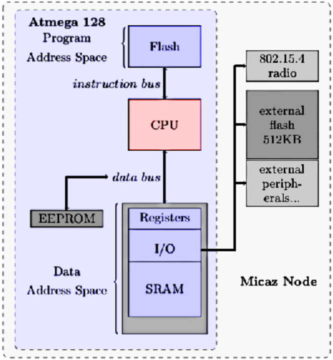
3 Our Approach

4 Risks, Outlook

5 Summary

6 References

Harvard architecture



[a]

Harvard architecture:

- Program (Flash) and Data Memory (SRAM) are physically separated
- Additional external memory (Flash, serial, slow)
- Program memory (128KB), read-only (except: boot loader)

[a] [2]

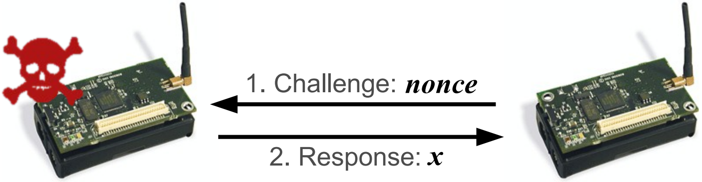
Adversary model

Adversary:

- Is able to inject bogus code $\widetilde{CI}$ into program memory
- Fully controls the node and its memories (program, data, external) until first round of attestation starts
- No control during attestation, but the node is already compromised possibly

→ How to detect injected code by using an attestation protocol?
→ How to detect program memory modifications?

Attestation using challenge-response protocols



Challenge-Response protocol:

- The base station sends a challenge *nonce* to a node
- Correct response $\implies$ the node is still trustworthy
- Wrong response $\implies$ the node has been compromised
- The base station knows the correct code image *CI* and owns precalculated challenge-response pairs $(nonce, x)$

Attestation using challenge-response protocols

Challenge-Response protocol:

- Calculation of response x: $x = h(nonce||CI)$
- *nonce* prevents replay attacks
- $h()$ is a hash- or checksumming-function

Some protocols:

- *nonce* acts as a seed for a PRNG
- CI's bytes are pushed into $h()$ in a pseudorandom manner

$\Longrightarrow$ We can detect whether the code image CI is manipulated or not

Noise filling

Problems:

- Only detects a manipulated CI, not a maniuplated program memory
- The size of CI ($|CI|$) is significantly smaller than the whole program memory usually (e.g. $128KB$)
- Unused program memory is filled with a constant value (e.g. 0xFF)
- The attacker is able to inject his $\widetilde{CI}$ into the empty program memory and to still pass the attestation

Idea: fill the empty program memory with pseudorandom data (PRW): noise filling [9]

$$x = h(nonce||CI||PRW) \tag{1}$$

Compression Attack

CCS'09 [3]:

- The attacker is able to compress CI using a lossless compression scheme (e.g. CHE), i.e. $C(CI)$
- Afterwards he can inject $\widetilde{CI}$ into the free space
- At attestation time he has to decompress $C(CI)$, i.e. calculate $C^{-1}(C(CI))$, on the fly
- The attacker can pass the attestation using such a compression attack, because the response x is correct

→ Our work starts here
→ Our Vision: Detection of compression attacks and design of a 'secure' attestation protocol

Outline

1 Introduction and motiviation

2 Code attestation protocols

3 Our Approach

4 Risks, Outlook

5 Summary

6 References

Description

Code Attestation with Compressed Instruction Code

Upload already compressed CI, i.e. $C(CI)$, to the node to not let the attacker compress $C(CI)$ to generate free space such easily.

$$x = h(nonce||C(CI)||PRW) \tag{2}$$

How does this scheme prevent compression attacks?

- The attacker gains no more free space from compressing $C(CI)$ (unlikely, not future-proof)
- Our compression raises the attacker's overhead 'by orders of magnitude', to let us detect him easily

Problems of our scheme

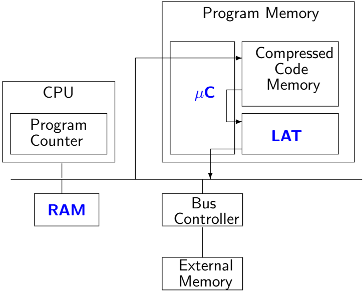
Problems:

- How to execute compressed code $C(CI)$?
- Which compression algorithm to use?
- The overhead of an on the fly decompression
- Attacker can generate better compression ratios possibly

Execution of compressed code

μC plus dictionary [8]

- No free choice of a compression algorithm

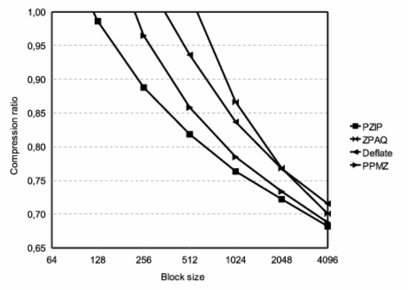


μC, Line Address Table (LAT) plus *Cache* [1] (1992)

- Block based compression
- The LAT knows the offsets of our compressed blocks
- The cache stores the current block (decompressed)
- Nearly free choice of a compression algorithm

$$x = h(nonce||C(CI)||PRW||LAT) \quad (3)$$

Choice of a compression algorithm



$$Ratio = \frac{C(CI)}{CI}$$

- *PPM*-Algorithms promise good compression ratios
- Larger blocks $\Longrightarrow$ better compression ratios
- *PZIP* promises best compression ratio (here)
- Better compression ratios achievable only through larger blocks or better compression algorithms

The attacker's overhead

We have to raise the attacker's overhead by orders of magnitude.
We benefit from our design:

- $x = h(...||C(CI)||...) \implies$ A 'clean' node does not have to decompress during the attestation
- The attacker has to decompress during the attestion to get back the original data and to calculate x correctly
- *nonce* as a seed for a PRNG, CI's bytes are pushed in a pseudorandom manner into $h()$

The attacker's overhead

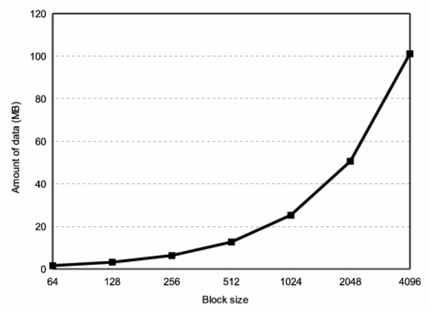
The attacker needs random access to $C(CI)$'s bytes:

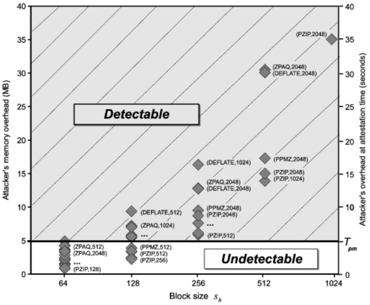
- The attacker does **not** have any random access
- The attacker has to use a LAT (i.e. LAT_a) and compresses blocks, too

Example: The attacker chooses a block size $s_a = 1024$ bytes

- The attacker has to decompress every of his compressed blocks 1024 times during the attestation
- In addition, he has to fetch every block 1024 times from the program memory

Memory overhead





Depending on the program memory's read performance, the number of blocks, i.e. the amount of free space for $\widetilde{CI}$ and the block size $s_a \implies$ The attacker's overhead increases

Outline

1 Introduction and motiviation

2 Code attestation protocols

3 Our Approach

4 Risks, Outlook

5 Summary

6 References

Open issues

Open issues:

- Work in Progress
- The overhead of our on the fly dekompresion is possibly not manageable
- Our scheme's complexity can possibly increase due to inevitable extensions
- The costs of a μC can be to high
- Possibly we'll never see a μC

Outlook

Next steps:

- Currently:
  - Proof of Concept Implementation in simulator (nearly done)
  - Determine attacker's overhead exactly
  - Determine timing approximately
  - Remaining problems (Attacks on the Cache)
- Port the scheme to other architectures? (e.g. Smartphones)

Summary

Summary:

- Low security of sensor nodes VS need for trustworthy sensor data
- Previous protocols: challenge-response, noise filling (PRW) ineffective [3]
- Our Protocol: Code Attestation with Compressed Instruction Code, $x = h(nonce||C(CI)||PRW||LAT_h)$
- We can prevent compression attacks [3]
- The attacker is not able to store a $\widetilde{CI}$ within the program memory and at the same time pass the attestation

References

- Wolfe, A., Chanin A.,
  Executing compressed programs on an embedded RISC architecture. ACM Sigmicro Newsletter, volume 23, pp. 81-91, (1992)
- Francillon, Aurélien and Castelluccia, Claude,
  Code injection attacks on harvard-architecture devices, CCS '08, 2008, Proceedings of the 15th ACM conference on Computer and communications security
- Claude Castelluccia, Aurélien Francillon, Daniele Perito and Claudio Soriente,
  On the Difficulty of Software-Based Attestation of Embedded Devices, ACM CCS 2009.

References

- Seshadri, A., Perrig, A., van Doorn, L., and Khosla, P. K.,
  SWATT: SoftWare-based ATTestation for embedded devices. In IEEE Symposium on Security and Privacy (2004), IEEE Computer Society.

- Seshadri, A., Luk, M., Perrig, A., van Doorn, L., and Khosla, P.,
  SCUBA: Secure code update by attestation in sensor networks. In WiSe 92,06: Proceedings of the 5th ACM workshop on Wireless security (2006), ACM.

- Shaneck, M., Mahadevan, K., Kher, V., and Kim,Y.,
  Remote software-based attestation for wireless sensors. In ESAS (2005).

References

Lefurgy, C., Bird, P., Chen, I., Mudge T.,
Improving Code Density Using Compression Techniques, Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, pp. 194-203, (1997).

H. Yamada, D. Fuji, Y. Nakatsuka, T. Hotta, K. Shimamura, T. Inuduka, T. Yamazaki,
Micro-Controller for reading out compressed instruction code and program memory for compressing instruction code and storing therein, US 6,98References

AbuHmed, T. and Nyamaa, N. and DaeHun Nyang,
Software-Based Remote Code Attestation in Wireless Sensor Network, Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE

References

- Abadi, M., Budiu, M., Erlingsson, U., and Ligatti J.,
  Control-flow integrity, In CCS'05: Proceedings of the 12th ACM conference on Computer and Communications Security (2005), ACM.
- Ferguson, C., Gu, Q., and Shi, H.,
  Self-healing control flow protection in sensor applications, In WiSec'09 (2009), ACM.
- Yang, Yi and Wang, Xinran and Zhu, Sencun and Cao, Guohong,
  Distributed Software-based Attestation for Node Compromise Detection in Sensor Networks, Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems

References

Huffman, D.A.,
A method for the construction of minimum redundancy codes.
Proceedings of the IRE 40 (1962)