

Code Attestation with Compressed Instruction Code

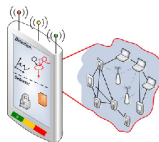
Benjamin Vetter (benjamin.vetter@haw-hamburg.de)
Prof. Dr. Dirk Westhoff (westhoff@informatik.haw-hamburg.de)

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik

Stand: 27. Januar 2011



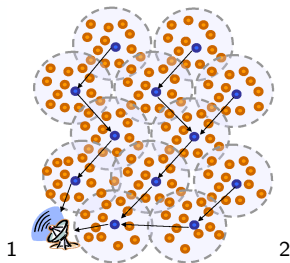
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences



Gliederung

- 1 Einführung und Motivation
- 2 Code Attestation Protokolle
- 3 Unser Lösungsansatz
- 4 Risiken, Abgrenzung, Ausblick
- 5 Zusammenfassung
- 6 Literatur

Mobile Sensorknoten



- Anwendungsgebiet: Monitoring, Detection, Control und Automation
- Vorteile: Flexibilität, geringe Kosten
- Nachteile: geringe Leistungsfähigkeit, geringe Sicherheit

¹<http://eccwsn.blogspot.com/>

²<http://www.informatik.uni-augsburg.de/>

Mobile Sensorknoten

Konflikt bzw. Motivation

Geringe Sicherheit \iff Abhängigkeit von verlässlichen
Sensordaten

Unsere Vision:

- 'Sicherere' mobile Sensorknoten
- Anwendung in kritischen Umgebungen (z.B. Health)



a

Idee: Code Attestation

Problem:

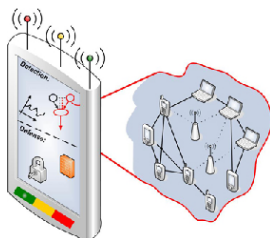
- Knoten 'leicht' kompromittierbar
- geringes Budget pro Knoten

⇒ keine teuren Trusted Platform Modules (TPMs) oder Tamper Proof Units einsetzbar

Idee: Code Attestation

Knoten muss einer vertrauenswürdigen Instanz beweisen, dass er noch vertrauenswürdig ist

Code Attestation im SKIMS-Kontext



3



4

SKIMS: mobile Geräte, Kooperativ, Mehrseitig

- Attestation: Leichtgewichtige Intrusion Detection
- Ergebnisse ggf. auch auf andere Architekturen portieren
- Kooperation in Health-Care-Szenario oder anderen sicherheitskritischen Umgebungen

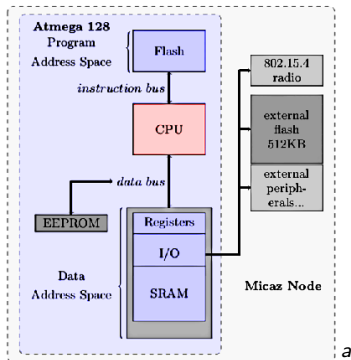
³<http://www.realmv6.org/skims.html>

⁴<http://eccwsn.blogspot.com/>

Gliederung

- 1 Einführung und Motivation
- 2 Code Attestation Protokolle**
- 3 Unser Lösungsansatz
- 4 Risiken, Abgrenzung, Ausblick
- 5 Zusammenfassung
- 6 Literatur

Harvard Architektur



Harvard-Architektur:

- Program (Flash) und Data Memory (SRAM) sind physikalisch getrennt
- Zusätzliches External Memory (Flash, seriell, langsam)
- Program Memory (128KB), Normalbetrieb: Read-Only, Bootloader: Read-Write

^a[2]

Adversary Model

Angreifer:

- Kann Bogus Code \widetilde{CI} in Program Memory einschleusen
- Hat vor der ersten Attestation vollständige Kontrolle über den Knoten und alle Memories (Program, Data, External Memory)
- Hat während der Attestation keine Kontrolle, aber der Knoten ist ggf. bereits kompromittiert und führt Schadcode aus

→ Wie kann die Attestation den eingeschleusten Code entdecken?

→ Wie können wir entdecken, dass das Program Memory modifiziert wurde?

Attestation mittels Challenge-Response Protokoll

Challenge-Response Protokoll:

- Die Basis-Station schickt einem Knoten eine Challenge *nonce*
- Korrekte Response \implies Der Knoten ist vertrauenswürdig
- Falsche Response \implies Der Knoten ist kompromittiert
- Die Basis-Station kennt das korrekte Code Image *CI*, das auf einem Knoten ausgeführt werden müsste und hat vorberechnete Challenge-Response Tupel (*nonce*, *x*)

Attestation mittels Challenge-Response Protokoll

Challenge-Response Protokoll:

- Die Response x berechnet sich z.B. so: $x = h(\textit{nonce} || CI)$
- *nonce* verhindert Replay-Angriffe
- Mit $h()$ als Hash- oder Checksum-Funktion

Einige Ansätze:

- *nonce* dient als Seed für einen Pseudozufallszahlengenerator
 - Die Bytes von CI fließen in pseudozufälliger Reihenfolge in $h()$
- ⇒ Wir erkennen, wenn Code Image CI manipuliert wurde

Noise-Filling

Probleme:

- Erkennt nur, dass CI manipuliert wurde
- Die Größe von CI (im weiteren $|CI|$) ist idR. wesentlich kleiner als das gesamte Program Memory (128KB)
- Nicht genutztes Program Memory ist idR. mit 0xFF gefüllt
- Der Angreifer kann \widetilde{CI} in den ungenutzten Speicher kopieren und die Attestation trotzdem bestehen

Idee: ungenutztes Program Memory mit pseudozufälligen Daten füllen (PRW): Noise-Filling [9]

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

Compression Attack

[3] auf der CCS '09:

- Kompression von CI mit verlustfreien Kompressionsverfahren durch den Angreifer (z.B. CHE), d.h. $C(CI)$
- Im so gewonnenen Program Memory kann der Angreifer \widetilde{CI} ablegen
- zur Attestation Zeit: On-the-fly Dekompression von $C(CI)$, d.h. $C^{-1}(C(CI))$
- Überlistet Code Attestation Protokolle, da die Response x korrekt ist

→ Hier fängt unsere/meine Arbeit an

→ Unsere Vision: Entdecken von Compression Attacks und Entwicklung eines 'sicheren' Attestation Protokolls

Gliederung

- 1 Einführung und Motivation
- 2 Code Attestation Protokolle
- 3 Unser Lösungsansatz**
- 4 Risiken, Abgrenzung, Ausblick
- 5 Zusammenfassung
- 6 Literatur

Beschreibung

Code Attestation with Compressed Instruction Code

Bereits komprimiertes CI , d.h. $C(CI)$, auf den Knoten laden, um zu verhindern, dass ein Angreifer CI wesentlich komprimieren kann

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

Wie kann dieser Ansatz eine Compression Attack verhindern?

- Der Angreifer kann $C(CI)$ gar nicht mehr gewinnbringend komprimieren (unwahrscheinlich, nicht zukunftssicher)
- Unsere Kompression erhöht den Aufwand für einen Angreifer 'by orders of magnitude', so dass der Overhead leicht zu entdecken ist

Probleme des Ansatzes

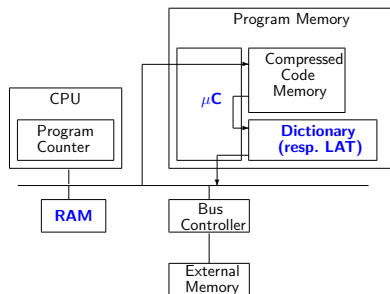
Probleme / Risiken:

- Das Ausführen von komprimiertem Code $C(CI)$
- Die Wahl des Kompressionsverfahrens
- Overhead durch on-the-fly Dekompression
- Der Angreifer kann ggf. 'besser' komprimieren

Ausführen von komprimiertem Code

μC und Dictionary [8]

- Beschränkte Wahl des Kompressionsverfahrens

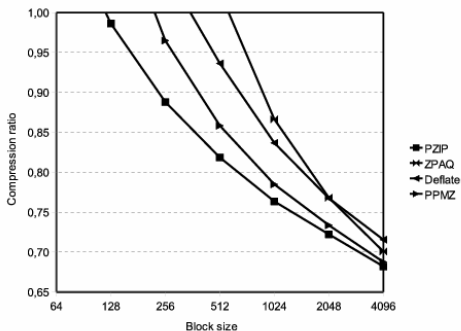


μC , Line Address Table (*LAT*) und *Cache* [1] (1992)

- Blockweise Kompression
- Die *LAT* besteht aus Startadressen komprimierter Blöcke
- *Cache* beinhaltet aktuellen dekomprimierten Code-Block
- Freie Wahl des Kompressionsverfahrens

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

Wahl des Kompressionsverfahrens



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

- *PPM*-Verfahren versprechen hohe Kompressionsraten
- Größere Blöcke \iff bessere Kompressionsraten
- *PZIP* erreicht beste Kompressionsraten

Overhead des Angreifers

Es geht darum, den Overhead des Angreifers deutlich zu erhöhen.
Uns kommt zu Gute:

- $x = h(\dots || C(Cl) || \dots) \implies$ Ein 'sauberer' Knoten muss während der Attestation nicht dekomprimieren
- Der Angreifer muss während der Attestation dekomprimieren um an die Originaldaten zu kommen und die Attestation zu bestehen, d.h. x korrekt zu berechnen
- *nonce* dient bei uns als Seed für einen Pseudozufallszahlengenerator, so dass die Bytes pseudozufällig in $h()$ fließen

Overhead des Angreifers

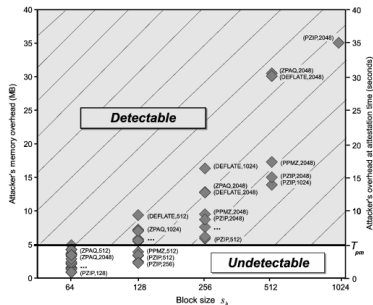
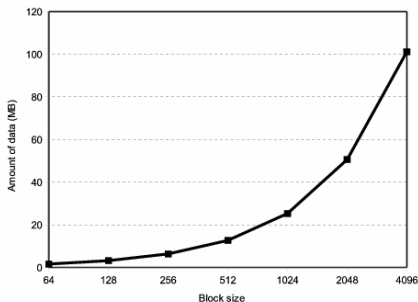
Der Angreifer benötigt 'wahlfreien' Zugriff auf $C(CI)$:

- Der Angreifer hat idR. **keinen** wahlfreien Zugriff
- Der Angreifer muss ebenfalls eine LAT verwenden (LAT_a) und blockweise komprimieren

Beispiel: Der Angreifer wählt die Blockgröße $s_a = 1024$ bytes

- Jeder vom Angreifer komprimierte Block muss vom ihm während der Attestation 1024 mal dekomprimiert werden
- D.h. der Block muss u.a. 1024 mal aus dem Program Memory gelesen werden

Memory-Overhead



Abhängig von der Geschwindigkeit des Program Memory, der Anzahl komprimierter Blöcke, d.h. der Menge notwendigen Speichers für \widetilde{CI} und der Blockgröße s_a wächst das Problem des Angreifers

Weitere Angriffe / Baustellen

Weitere Angriffe / Baustellen:

- **Wie groß kann \widetilde{CI} maximal sein**
- **Angriffe auf den Cache**
- Denial of Service (DoS)
- Angriffe auf LAT_h
- Data Memory
- External Memory
- Replay-Angriffe
- Node Depletion

Gliederung

- 1 Einführung und Motivation
- 2 Code Attestation Protokolle
- 3 Unser Lösungsansatz
- 4 Risiken, Abgrenzung, Ausblick**
- 5 Zusammenfassung
- 6 Literatur

Verbleibende Risiken

Verbleibende Risiken:

- Unser Ansatz erweist sich doch als ineffektiv
- Der Overhead der on-the-fly Dekompression ist zu hoch
- Die Komplexität des Verfahrens wird sehr groß
- Die Kosten für einen HW-Support sind zu hoch
- Es ist kein HW-Support in Aussicht
- Eine Portierung des Ansatzes ist nicht möglich

Abgrenzung

Abgrenzung:

- Die Sicherheitsaspekte stehen im Fokus
- Synergieeffekte durch Code-Kompression 'nur' Nebeneffekt, der jedoch **für** den Ansatz spricht
- Dennoch ist kein Hochsicherheitsniveau möglich, da ein Angreifer die Hardware manipulieren kann

Ausblick

Wo wollen wir hin:

- Verbleibende Probleme lösen
- Proof of Concept Implementierung
- Wir benötigen Sensorknoten und einen konkreten μC
- Konkrete Bestimmung unseres Overheads

SKIMS:

- Den Ansatz ggf. auf andere Architekturen zu portieren
- Kooperation von Sensorknoten und anderen mobilen Geräten in sicherheitskritischem Umfeld (z.B. Health)

Zusammenfassung

Zusammenfassung:

- Schwache Sicherheit von Sensor-Knoten \iff Bedarf nach vertrauenswürdigen Sensordaten
- Bisheriger Ansatz: Challenge-Response Protokoll, Noise-Filling (*PRW*) aufgrund von [3] unwirksam
- Unser Ansatz: Code Attestation with Compressed Instruction Code, $x = h(\text{nonce} || C(CI) || PRW || LAT_h)$
- Wir glauben, unser Ansatz kann Compression Attacks [3] verhindern
- Der Angreifer kann \widetilde{CI} nicht unbemerkt ins Program Memory einschleusen

Literatur



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.

Literatur

-  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).

Literatur



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 6,986,029 B2



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

Literatur

-  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

Literatur



Huffman, D.A.,

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