Secure and Efficient Software-based Attestation for Industrial Control Devices with ARM Processors

Binbin Chen*, Xinshu Dong*, Guangdong Bai#,
Sumeet Jauhar*, Yueqiang Cheng^

* Advanced Digital Sciences Center, Illinois at Singapore
# Singapore Institute of Technology
^ Baidu USA XLab

ACSAC 2017, December 7, 2017, Orlando FL, USA
Attacks on Industrial Control Systems

2015 Ukraine Power Grid Attack:
- Attacker abuses the SCADA and field devices to open circuit breakers
- The attackers persist within the environment for six months or more
- Serial-to-Ethernet communications devices impacted at a firmware level
- Lack of active defense measures

2016 Ukraine Power Grid Attack:
- Automated with a malware framework

TLP: White
Analysis of the Cyber Attack on the Ukrainian Power Grid Defense Use Case

'Crash Override': The Malware That Took Down a Power Grid
Software Attestation for ICS Devices

• Ensuring the software integrity of ICS devices is a foundational requirement for enhancing ICS security.
Software Attestation for ICS Devices

• ICS devices execute well-specified program logic. This allows an attestation-based approach to provide high assurance about the software integrity of the attested device

• In comparison, solutions like anti-virus or host-based intrusion detection systems only provide best-effort malware detection
SoftWare-Only Root of Trust (SWORT)

• Attestation requires some form of root of trust
  – It is often difficult to deploy a hardware-based root-of-trust solution (e.g., TPM) to existing ICS
  – A software-only solution hence is desirable

Verifier (Trusted)
1. Generate random nonce
5. Verification of checksum value & response time

2. Checksum request with nonce

4. Checksum result

Prover (Untrusted)
3. Checksum computation
6. Hash computation (optional)
Efficient SWORT for ICS

• SWORT consumes all computational resources of the prover, hence it needs to run fast for ICS
  – E.g., the cycle response time is 20ms for a 50-Hz grid

• Short attestation time also helps raise the bar for launching proxy attacks, where the device under attestation asks a computationally more powerful remote device to compute the checksum
  – E.g., if SWORT completes within 20ms, a proxy attack behind a slow link (e.g., cellular) would fail
Full Memory Walk May Take Too Long

• For example, consider an NXP LPC2362 board with an ARM7 72MHz CPU and a 58KB RAM, and a probability of $\Pr[\text{win}] = 10^{-10}$ for a prover with modified memory to cheat the verifier and win the attestation, a random walk over the whole RAM takes 156.9ms.

$$N = s \times \ln \left( \frac{1}{\Pr[\text{win}]} \right)$$

• If RAM size grows to a few hundred MB, the memory walk can take minutes to complete
Outline

• Background

• Memory Stride Design

• Analysis and Evaluation
System & Threat Model

• We focus on ICS devices with low-end ARM processors, e.g., ARM7 or ARM Cortex-M3
• We assume there is a trusted verifier that locates in the same local area network (LAN) as the proofer
  – The communication delay between the verifier and the proofer is short and stable
• We trust the physical access control to the ICS environment
  – No proxy devices inside the LAN, no change to proofer’s hardware spec
  – Malware can still be introduced, though, e.g., by innocent insider during maintenance
• We do not address malware that launches attacks outright (e.g., DoS attacks)
Attacks to Partial Memory Walk

- Prior secure partial memory walk solutions
  - Pioneer for Intel platform
  - SCUBA & ICE scheme for a MIPS platform
  - Challenges on ARM platform have been discussed, but no concrete solution has been proposed

Illustration of Memory Copy Attacks
Attacker’s Potential Leverage on ARM

• “Free” Offset/Shift with LDR
  – E.g., \texttt{LDR r0, r1, \#0x1000} takes the same time as \texttt{LDR r0, r1}
    where 0x1000 is the offset of a clean copy to launch attack in (b)

• “Free” ARM-Friendly Immediate Value
  – E.g., one can hard code the PC value to launch attack in (c)
Memory stride

• Basic idea I: Two types of memory accesses
  – Memory walk over SWORT code (size \( L \))
  – Memory stride over stride addresses (neighboring stride addresses are separated by \( L \). Hence for a RAM size of \( s \), there are \( \frac{s}{L} \) stride addresses (see \( x_0, x_1, x_2, x_3 \) …)
  – The total number of addresses to cover reduces from \( s \) to \( L + \frac{s}{L} \)
Memory stride

• Basic idea II: Access these two types of memory addresses in an interleaving way
• Basic idea III: Generate unique values for stride addresses (except for $x_0$)
How it works

• No matter how the attacker moves the code region or the stride addresses, there will always be at least one overlapping word between these two address spaces.
• This collided address supposes to return different values for the two different types of accesses.
Outline

• Background

• Memory Stride Design

• Analysis and Evaluation
We follow the ASSW analysis framework and simplify their upper bound results based on typically used values of parameters.

\[
Pr[Win] \leq \frac{p + s}{l_r / l_s} \cdot 2^{-(l_g + l_r)} + \max\{\omega, \nu_{Chk}\} + \max_{0 \leq M \leq N}\left\{\pi(M, \text{ops}) + \varrho \right\} \cdot \gamma^{N-M} + \nu_{Gen} \cdot (N - M)\]

where:
\[
\pi(n, x) := \sum_{j=\max\{0, n - 2^{l_a}\}}^{n-1} \left(\max\{\lambda^{x+1}, \gamma\}\right)^{n-j} \cdot \binom{n}{j} \cdot \left(\prod_{i=0}^{j} \frac{2^{l_a} - i}{2^{l_a}}\right) \cdot \left(\frac{n-j}{2^{l_a}}\right)^{j}
\]

We simplify it to:
\[
Pr[Win] \leq \pi(N, \text{ops})
\]

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Representative Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p)</td>
<td>primary memory size (in words)</td>
<td>16 (directly accessible registers)</td>
</tr>
<tr>
<td>(s)</td>
<td>secondary memory size (in words)</td>
<td>4K words for a 16KByte memory</td>
</tr>
<tr>
<td>(l_a)</td>
<td>length of effective secondary memory address (in bits)</td>
<td>(s = 2^{l_a})</td>
</tr>
<tr>
<td>(\lambda)</td>
<td>the fraction of memory words that are identical in memory state 5 and 5</td>
<td>(1 - \frac{1}{2^{l_a}})</td>
</tr>
<tr>
<td>(l_g)</td>
<td>length of pseudo random seed, in bits</td>
<td>32</td>
</tr>
<tr>
<td>(l_r)</td>
<td>length of the checksum, in bits</td>
<td>(32 \times 12)</td>
</tr>
<tr>
<td>(l_s)</td>
<td>length of a state entry (i.e., a memory word), in bits</td>
<td>32</td>
</tr>
<tr>
<td>(\Sigma)</td>
<td>the set of possible state entries (memory words)</td>
<td>(O(2^{l_a}))</td>
</tr>
<tr>
<td>(\gamma)</td>
<td>state-incompressibility parameter</td>
<td>(\max_{x \in \Sigma} D_{\gamma}(x))</td>
</tr>
<tr>
<td>(N)</td>
<td>number of iterations for memory accesses</td>
<td>(O(s)) (see Equation 5)</td>
</tr>
<tr>
<td>(\varrho)</td>
<td>time-bounded pseudo-randomness of Gen</td>
<td>0</td>
</tr>
<tr>
<td>(\nu_{Gen})</td>
<td>time-bounded unpredictability of Gen</td>
<td>0</td>
</tr>
<tr>
<td>(\nu_{Chk})</td>
<td>time-bounded unpredictability of Chk (N)</td>
<td>(\frac{1}{2^{l_r}})</td>
</tr>
<tr>
<td>(\omega)</td>
<td>Blind second pre-image resistance for Chk</td>
<td>(\frac{1}{2^{l_r}})</td>
</tr>
</tbody>
</table>

Table 2: Main notations used in the ASSW framework.

An Issue of ASSW Framework

• The ASSW framework makes two strong and pessimistic assumptions:
  – Once an iteration accesses a collision address, the attacker has 100% chance to win, i.e., compute the correct checksum for that iteration while using 0 time
  – The attacker has 100% chance to win one iteration, if the attacker spends (ops + 1) (instead of ops) time for one iteration

• Our analysis shows, regardless of the value of $N$, when the similarity between malicious image and genuine image is high, i.e., $\lambda = 1 - \frac{1}{2^l}a$, a simple attack strategy can achieve

$$\Pr[\text{win}] > \frac{1}{e^{1+\frac{1}{ops}}}$$

• Conclusion: we cannot keep both assumptions
  – ASSW paper didn’t realize this issue, partly because they evaluate a smaller value of $\lambda$
Proposed Change to ASSW Framework

• Drop ASSW framework’s pessimistic assumption about collision addresses
  – There is no known attack that can “remember” previously accessed address
• With this change, we are able to further simplify the upper bound to: \( \Pr[\text{win}] < (1 - 2^{la})^N \), hence,
  \[ N = 2^{la} \times \ln\left(\frac{1}{\Pr[\text{win}]}\right) \]
• Applying this result, our memory stride solution reduces the time for attestation from \( O(s) \) to \( O\left(L + \frac{s}{L}\right)\)
Evaluation

- We implemented memory stride on two RTU models
  - Memory stride can always complete SWORT within 20ms
  - Performance gain > 10x compared to full RAM walk in several settings

<table>
<thead>
<tr>
<th>Device</th>
<th>Setting</th>
<th>10 nines assurance</th>
<th>5 nines assurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC2292</td>
<td>ARM7 (60MHz, 16KB RAM)</td>
<td>52.1ms</td>
<td>26.1ms</td>
</tr>
<tr>
<td>LPC2362</td>
<td>ARM7 (72MHz, 58KB RAM)</td>
<td>156.9ms</td>
<td>78.5ms</td>
</tr>
<tr>
<td>LPC1756</td>
<td>Cortex M3 (100MHz, 16KB RAM)</td>
<td>27.6ms</td>
<td>13.9ms</td>
</tr>
<tr>
<td>LPC1788</td>
<td>Cortex M3 (120MHz, 96KB RAM)</td>
<td>137.0ms</td>
<td>68.6ms</td>
</tr>
</tbody>
</table>
Impact of Network Delay

• We evaluated two models of industrial Ethernet switches, Belden Spider II 8TX Ethernet switch and Moxa EDS-205 Switch
• First 5 hops are Belden switches and next 5 are Moxa switches
Impact of Network Delay

- If checksum computation is 12.5 ms, and assume a malicious prover incurs x% of overhead, (a pessimistic setting of x=1.6%)
  - 12.5ms x 1.6% = 0.2ms
  - Even a single hop network delay is greater than 0.2ms, so network delay needs to be explicitly accommodated
Impact of Network Delay

- Consider the per-hop minimum delay is already factored in. For a given hop, we find that the probability for the delay to be 0.2ms above the minimum delay is less than 0.1%. Hence, with a delay budget of 0.2 ms, the false positive rate can be low.
Conclusion

• A new *memory stride* design to reduce the SWORT time requirement on ARM-based devices

• Analysis based on an adapted version of ASSW framework
  – The proposed change is needed for practical application of the framework for SWORT analysis (not only for memory stride)

• A push towards real-world system integration

**Thanks!**

Contact:  binbin.chen@adsc.com.sg