PowerAlert: An Integrity Checker using Power Measurement

Ahmed Fawaz
11/16/2016
Submitted to IEEE S&P’17
Problem Description

How to validate the integrity of software against a slow and stealthy attacker without any trusted components in the machine?

• Trusting data from monitors such as Kobra
Solution Approach

Fig. 1: The components of PowerAlert.
Threat Model

- Host machine is untrusted with attacker changing the kernel (rootkit)
- Attacker does not modify the hardware (clock rate, firmware,...)
- Attacker will attempt to deceive PowerAlert

<table>
<thead>
<tr>
<th>Proxy Attack</th>
<th>Data Pointer Redirection</th>
<th>Static Analysis</th>
<th>Active Analysis</th>
<th>Attacker Hiding</th>
<th>Force Retraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a remote machine to compute</td>
<td>Keep a clean copy of memory</td>
<td>Attempt to precompute</td>
<td>Learn how the program works</td>
<td>Hide when the checker runs</td>
<td>Learn a tampered model</td>
</tr>
</tbody>
</table>
Solution

• Diversity of the IC-program to thwart adaptation
• Measure current independently
• Run the checker at random times
PowerAlert Protocol

Verifier:
1. Start protocol
2. Generate the IC-program
3. Start measurement
4. Verify the output and the current measurements

Prover:
1. Pause all tasks
2. Load the IC-program
3. Run the IC-program
4. Return the result

Fig. 2: Description of the POWERALERT-protocol
Generation of IC-programs

Fig. 4: Control Structure

Fig. 3: General Architecture of the Hash Function
Algorithm V.2 Irreducible polynomial generation using Ben-Or Irreducibility Test

while true do
  Generate poly \( p(x) \in GF(2) \) of degree at most \( n \)
  for \( i := 1 \) to \( n/2 \) do
    \( g := \gcd(p, x^{2^i} - x \mod p) \);
    if \( g \neq 1 \) then
      '\( p \) is reducible'
      break
    end if
  end for
  return '\( p \) is irreducible'
end while
Power Finite
State Machine
Power Measurement

Fig. 6: Block diagram for power state extraction.

Fig. 5: The current measurement loop place around the CPU power line.
Measurements
Timing Model

IC-program runtime:
\[
x = [N, \|f\|_c]
\]
\[
y = 1.3958 + 0.081x(1) - 0.017x(2) + 0.008x(1) \times x(2)
\]
\[
\sigma = 5.4542 \mu s
\]

Network runtime:
\[
y_n = 0.129 \times x + 12.48
\]
\[
\sigma_n = 1.902 \mu s
\]

Fig. 9: Power State Timing model for hashing phase.
Verification Process

• Power Language:

\[ L = (S_0, S_1)^+ (S_0, S_2, S_3, S_0)(S_0, S_1) \]

• Timing verification:

\[ \delta \geq \gamma \max(\sigma_m + \sigma_s), \quad \delta_n \geq \gamma \max(\sigma_n, \sigma_s) \]
Example

Fig. 10: Timing difference in current signal due to tampering.
Design of IC-program

\[
\begin{align*}
\text{minimize} & \quad y(N, \|f\|_p) \\
\text{subject to} & \quad y(N, \|f\|_p + k) - y(N, \|f\|_p) > \gamma \cdot \max(\sigma_m, \sigma_s) \\
& \quad y_n(\|f\|_p) > \gamma \cdot \max(\sigma_n, \sigma_s) \\
& \quad \|f\|_p < \text{cost, } N/N_{total} > \text{coverage.}
\end{align*}
\]
Attacker-Verifier Games

• Players:
  • Attacker
  • Defender

• Actions:
  • Attacker hide @ \{ta_1, ta_2, ta_n, \ldots\}
  • Defender verify @ \{td_1, td_2, td_n, \ldots\}

• Strategy: pick the time instances
Strategy selection

• Defender is assumed to have a renewal strategy with rate $\lambda$

**Theorem 1.** If the verifier is playing with a renewal strategy, then the attacker’s best strategy is to play periodically with a fixed period $T^*$. 
Detection probability

![Graph](image)

Fig. 11: The average probability of the attacker’s detection as a function of the attacker’s and the verifier’s play rates.
Fig. 12: The average fraction of the time the attacker’s malicious activity is hidden as a function of the attacker’s and the verifier’s play rates.
Fig. 13: The average ratio of attestation tasks that were evaded by the attacker’s actions as a function of the attacker’s and the verifier’s play rates.
Fig. 14: Average time in seconds for generating a random irreducible polynomial for degree $d$

Fig. 15: Maximum Number of IC-Programs a varying tree depth and polynomial degree.
## Security Analysis

<table>
<thead>
<tr>
<th>Proxy Attack</th>
<th>Data Pointer Redirection</th>
<th>Static Analysis</th>
<th>Active Analysis</th>
<th>Attacker Hiding</th>
<th>Force Retraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a remote machine to compute</td>
<td>Keep a clean copy of memory</td>
<td>Attempt to precompute</td>
<td>Learn how the program works</td>
<td>Hide when the checker runs</td>
<td>Learn a tampered model</td>
</tr>
<tr>
<td><strong>Network timing will detect redirection</strong></td>
<td><strong>IC-program optimized to detect k=4</strong></td>
<td><strong>The structure is flattened thus NP-Hard</strong></td>
<td>• Change of the IC-program prevents adaptation.</td>
<td>Defender strategy leads to attacker slow down</td>
<td>Retraining is only done with a clean HDD</td>
</tr>
</tbody>
</table>

- Network timing will detect redirection: IC-program optimized to detect $k=4$.
- Static Analysis: The structure is flattened thus NP-Hard.
- Active Analysis: • Change of the IC-program prevents adaptation. • Space of programs big.
- Attacker Hiding: Defender strategy leads to attacker slow down.
- Force Retraining: Retraining is only done with a clean HDD.
Conclusion

• We use power measurements as a trust base

• The IC-program diversity prevents the attacker from adapting

• The defender's strategy results in the attacker detection