A Mutual Auditing Framework to Protect IoT against Hardware Trojans

Chen Liu, Patrick Cronin, and Chengmo Yang

10/03/2016
Outline

• Hardware Trojan in IoT
• Proposed Trojan detection scheme
• Simulation results
• Summary
**Hardware Trojan**: malicious elements inserted in circuit

- IP house
- Design house
- SoC integrator
- Layout (GDSII)
- Fabrication
- IC
- IC testing & Deployment
- User

- USA
- Design house

- Europe

- Asia
- Foundry

- Customer

- 3PIP vendor
Hardware Trojan: malicious elements inserted in circuit

Unwanted malicious circuit

Difficult to detect during testing:
1. Inserted at hard-to-detect place
2. May lack of golden model
3. May be hibernated

Product with hardware Trojan sold to the customer...
A Hardware Trojan may...
tamper output

Device with Trojan

Output → Trojan → Wrong Output

send secret message

Device with Trojan

Output → Trojan → Secret output
Hardware Trojans in a network may...
Internet of Things (IoT)

Trojans?

Wireless communication

Nodes

Servers
An active Trojan may trigger the hibernated ones by sending triggering messages. In normal operation, the output in the trigger is 1000. If the trigger is 1000, the MUX output is also 1000. Entire network down in a short while. Fault tolerance does not work. Catastrophe.
Problem to solve: **hardware Trojan collusion in IoT**

Previous hardware Trojan countermeasures \([1,2,3]\) may detect single Trojan but not **Trojan collusion**

Previous IoT security solutions target attacks from outside of the network \([4,5,6]\) but not **attacks from the inside**

Our goal: prevent hardware Trojan in IoT from mutually triggering

Outline

• Hardware Trojan in IoT
• Proposed Trojan detection scheme
• Simulation results
• Summary
Our goal: prevent hardware Trojan in IoT from mutually triggering

Our method:

- Message encryption
- Mutual auditing
- Vendor diversity
Each node is assigned a unique cryptography key.

Each message should be encrypted with symmetric encryption.

Cryptography shuffles message, including the Trojan trigger.

Encryption to shuffle Trojan trigger

Cannot decrypt!
Have Trojan?

Y

Encrypt the message?

N

Send trigger in plaintext

Y

With correct key

N

Send trigger with wrong encryption key

Y

Send trigger with correct encryption key

How to ensure correct encryption?

Message tampered by routing nodes

Send trigger with correct encryption key

Trojan trigger being shuffled

HOWEVER, encryption by itself cannot fully solve the problem!
Let’s introduce Mutual auditing
Node mutual auditing

**First-hop auditing:** each node is audited by its neighbor nodes

**Echo auditing:** each auditor node is also audited by the node before
Node mutual auditing

**First-hop auditing:** each node is audited by its neighbor nodes

**Message sender:** insert pre-defined audit bits

- Header
- Body
- Audit bit ≤ 128 bits

**Auditor:** check if the audit bits are valid

- Header
- Encrypted Body
- FCS

For 128-bit AES, every block is encrypted.

**Echo auditing:** each auditor node is also audited by the message sender

- Frame check sequence – for fault tolerance

- Valid
Node mutual auditing

**First-hop auditing:** each node is audited by its neighbor nodes

Message sender: insert pre-defined audit bits

- Header 1 0 1 1
- Body
- Audit bit
- ≤128 bits

Auditor: check if the audit bits are valid

- Header
- Encrypted Body
- FCS
- Frame check sequence – for fault tolerance

Echo auditing: each auditor node is also audited by the message sender
Node mutual auditing

**First-hop auditing:** each node is audited by its neighbor nodes

**Echo auditing:** each auditor node is also audited by the message sender
Security analysis for a node

- Have Trojan?
  - N: All audits passed
  - Y: Auditee secure?
    - N: Tamper the message audited
      - Fails echo auditing
    - Y: Encrypt the message?
      - N: Send trigger in plaintext
        - Fails first-hop auditing
      - Y: With correct key?
        - N: Send trigger with wrong encryption key
          - Fails first-hop auditing
        - Y: Send trigger with correct encryption key
          - Trigger being shuffled
          - No false positive
Overhead analysis

Regular IoT with message encryption:

1 × encryption

\[ N_0 \rightarrow N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_k \rightarrow \text{Server} \]

Message generator \quad k \text{ hops} \quad \text{Server}

Proposed scheme:

1 × encryption

First-hop auditing

1 × decryption

Echo auditing

\[ k \times \text{comparison} \]

\[ N_0 \rightarrow N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_k \rightarrow \text{Server} \]

Message generator \quad k \text{ hops} \quad \text{Server}

In parallel with message forwarding, does not add any delay
How to prevent auditor and auditee from collusion?
Node vendor diversity

Different vendors
Different triggers
Unable to mutually trigger
Node vendor diversity – how many vendors?

One vendor per node = 100% secure  = huge overhead
Node vendor diversity – how many vendors?

Color of auditee ≠ Color of auditor → Secure

Determine 8 vendors → 3 vendors coloring

Node routing map
Graph coloring algorithm
Node vendor selection
Outline

• Hardware Trojan in IoT
• Proposed Trojan detection scheme
  – Message encryption
  – Mutual auditing
  – Vendor diversity
• Simulation results
• Summary
# Methodology

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation tool</td>
<td>NS-2</td>
</tr>
<tr>
<td>Network scale</td>
<td></td>
</tr>
<tr>
<td>network size</td>
<td>10 × 10 to 20 × 20</td>
</tr>
<tr>
<td>max bandwidth</td>
<td>100 MB/s</td>
</tr>
<tr>
<td>expected traffic</td>
<td>40 to 100 packets/s</td>
</tr>
<tr>
<td>Network parameters</td>
<td></td>
</tr>
<tr>
<td>packet size</td>
<td>200 B body + 78 B metadata</td>
</tr>
<tr>
<td>packet processing time</td>
<td>1 ms per hop</td>
</tr>
<tr>
<td>cryptography overhead</td>
<td>1 ms per 128 bits</td>
</tr>
</tbody>
</table>
Security study by simulating Trojan activation

A hibernated Trojan can be either:

- **Self triggered** with a probability of $p$ per packet
- **Mutually triggered** by successfully receiving and decoding triggering message sent by active Trojan from the same vendor

In the baseline, almost all the Trojans are activated.

With the proposed scheme, mutually triggering is eliminated.

With larger $p$, more Trojans are self-triggered.
Performance evaluation

Latency vs Network Size

- latency increases due to more hops per packet

Completion Rate vs Network Size

- lower completion rate due to more hops per packet

Throughput vs Network Size

- slightly lower throughput, since packets are more prone to be dropped

With larger network size..
Performance evaluation

Latency vs Network Size

Completed Rate vs Network Size

Throughput vs Network Size

With the proposed scheme...

- introduces constant latency (~25ms), due to the overhead of encryption/decryption.
- introduces almost negligible impact on the completion rate.
- introduces almost negligible impact on the throughput.

Network Size

Latency (ms)

Completion Rate (%)

Throughput (B/s)
Outline

- Hardware Trojan in IoT
- Proposed Trojan detection scheme
- Simulation results
- Summary
Summary

• Problem:
  • Hardware Trojans are malicious and covert changes to the circuits which are difficult to detect during testing.
  • In IoT, hardware Trojans in different nodes may mutually trigger each other to cause catastrophe.

• Proposed framework:
  • Goal: prevent hardware Trojans in IoT from mutually triggering.
  • Method combines:
    • message encryption
    • node mutual auditing
    • node vendor diversity

• Simulation results show that the proposed scheme:
  • Prevents hardware Trojans from mutually triggering each other.
  • Introduces a constant (~25ms) latency to each packet regardless of the network size and traffic volume.