CERIAS

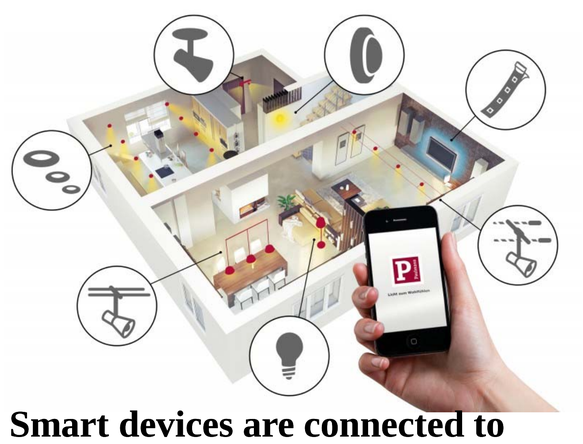
The Center for Education and Research in Information Assurance and Security

An Automated and Principled Security Analysis Framework for Bluetooth LE Implementations

Syed Rafiul Hussain*, Victoria C. Moore †, Elisa Bertino*

*Purdue University, † Intel Corporation

Bluetooth Low Energy For Proximity-based Communication



Smart devices are connected to IoT gateways, e.g., Smartphone

Vulnerabilities in Bluetooth Protocol Implementation

Nest security cameras can be knocked out via Bluetooth

The Washington Post
Democracy Dies in Darkness

Innovations

What we know about car hacking, the CIA and those WikiLeaks claims

Jasek said the problem is traced back to devices that use the Bluetooth Low Energy (BLE) feature for access control. He said too often companies do not correctly implement the bonding and encryption protections offered in the standard. This shortcoming could allow attackers to clone BLE devices and gain unauthorized access to a physical asset when a smartphone is used as a device controller.

Why Such Vulnerabilities?

(1) Parsing Errors: BLE Implementations do not correctly parse and process the BLE packets.

(2) Semantic Bugs: Implementations deviate from Bluetooth standard specifications and hence contains functional or semantic bugs.

(3) Memory Corruption Bugs: Use-after-free, buffer overflow, etc.

(4) Weak Cryptographic Primitives: Cryptographic building blocks used in the protocol are prone to existing attacks.

Problem Objective

Why Existing Techniques Fall short?

Fuzzing:

- Cannot explore the functional bugs.
- Cannot point out the location of the bug.
- Poor code coverage

Symbolic Execution:

- State explosion problem.

Problem Statement

Develop a highly automated security evaluation framework to detect first two types of bugs.

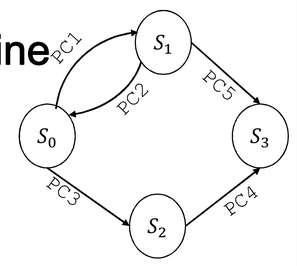
Our Proposed Approach

(1) Extract Finite State Machine

- Using a combination of
  - ✓ static analysis
  - ✓ symbolic execution

(2) Security Evaluation

- Find missing checks
- Use model checking to find property violation
- Perform differential testing by comparing two FSMs



Extract FSM

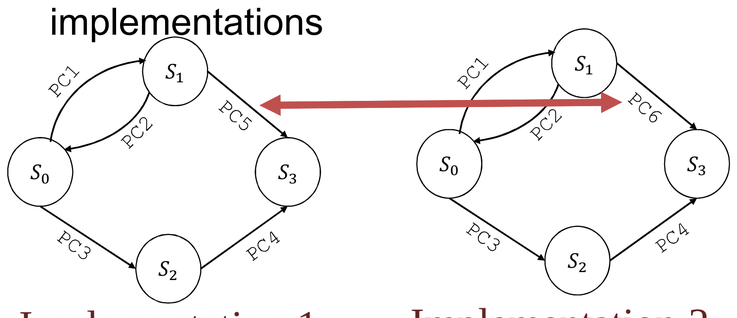
- Slice the required portion → Build Control Flow Graph → Perform Points-to Analysis → Construct Call Graph
- Find State Variables → Find Global Variables → Find variables that are updated while processing packet
- Symbolic Execution → Extract Path Constraints → Use Path Constraints as Transitioning Conditions in FSM

Find Missing Checks

Malicious packets may get accepted by an implementation if certain checks are missed

Solutions:

1. Compare path constraints for two different implementations



Implementation 1 Implementation 2

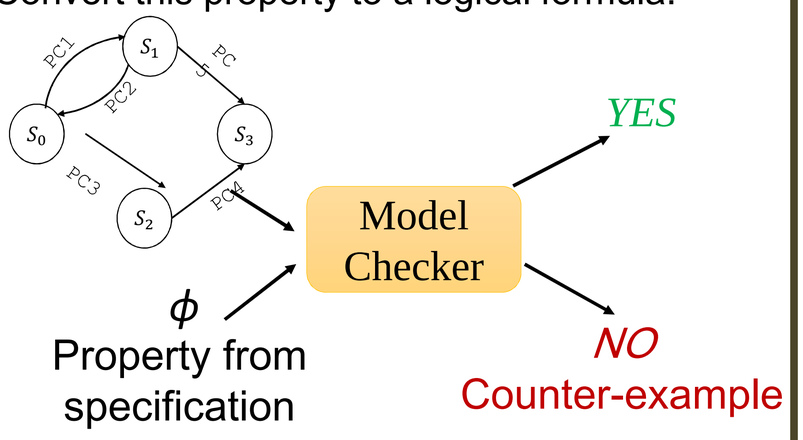
2. Find the relevant fields of a packet in the list of path constraints.

Find Property Violation

- Select important security property from standard specification.

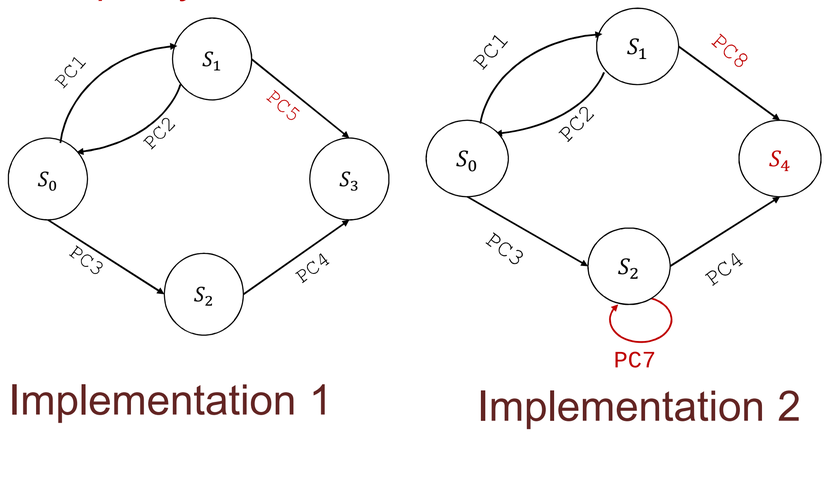
"The length of the pin code must not exceed 128 bits"

- Convert this property to a logical formula.



Differential Testing

Difference between two FSMs refers to possible discrepancy



Implementation 1 Implementation 2

Porsche's Car Kit Authentication

Car Kit's authentication bypass with Android Phone

Goes directly to BLE_PAIR_AUTH_COMPLETE state if there is a saved PIN code.

Overflow PIN Code Memory in BlueDroid

If a malicious client sets a pin that was too long it would overflow the pin code memory.

```
bt_status_t btif_dm_pin_reply( … ){
        …
        - if (pin_code == NULL)
        -       return BT_STATUS_FAIL;
        + if (pin_code == NULL || pin_len >
PIN_CODE_LEN)
        +       return BT_STATUS_FAIL;
#if (defined(BLE_INCLUDED) && (BLE_INCLUDED ==
TRUE))
```

Conclusion

Though, developers often optimize the complex part of the specification for embedded devices, they need to make sure the implementation complies with specification.

*This work is supported by Intel Corporation

CERIAS

PURDUE UNIVERSITY