LIGHTBLUE: Automatic Profile-Aware Debloating of Bluetooth Stacks

Jianliang Wu¹, Ruoyu Wu¹, Daniele Antonioli², Mathias Payer², Nils Ole Tippenhauer³, Dongyan Xu¹, Dave (Jing) Tian¹, Antonio Bianchi¹

¹ Purdue University ² EPFL ³ CISPA
Background

• Bluetooth devices are everywhere
• Support different use cases
  ▪ Audio streaming
  ▪ Printing
  ▪ Smart home
  ▪ Health care
Motivation - Bluetooth is bloated

- **Multiple profiles**
  - GATT (Generic Attribute Profile)
  - PAN (Personal Area Network), etc.
- **Diverse protocols**
  - L2CAP, BNEP, etc.
- **Two components**
  - Host code
  - Firmware
Motivation - Example

Android phone

Square app

Credit card

Square credit card reader

Bluetooth stack

- MAP
- PAN
- HID
- GATT
- SDP
- RFCOMM
- BNEP
- ATT
- L2CAP
- HCI
- Firmware

Square app
Objectives

• Debloating unneeded code so that the vulnerabilities within the unneeded code are no longer exploitable.

• Automatic
  ▪ LightBlue should automatically remove unneeded code

• Flexible
  ▪ LightBlue should support debloating different profiles

• Full stack debloating
  ▪ Debloating across different components (host code and firmware)
1. Identify needed profiles

2. Identify and remove code not used by the needed profile in the host (host code debloating)

3. Identify and remove unused HCI command handlers in the firmware (firmware debloating)
Step 1. Profile identification

• Scan for APIs used to invoke a profile’s functionalities
  ▪ E.g., getProfileProxy() on Android
Step 2. Host code debloating (source code)

- Profile aware dependence analysis
  - Profiles might be coupled
    - Per-profile data analysis
Step 2. Host code debloating (source code) cont.

- Profile aware dependence analysis (cont.)
  - One profile has multiple entry functions
    - Data analysis cannot directly apply
    - Transform multiple-entry interface into a single-entry interface
      - Divide profile entries into 3 categories (initiating, functioning, and ending)
      - Create a dummy function mimicking the profile life-cycle

- Code removal and HCI commands extraction

```c
static const btav_interface_t bt_av_src_interface = {
    sizeof(btav_interface_t),
    init_src,
    src_connect_sink,
    disconnect,
    cleanup_src,
};
```
Step 3. Firmware debloating (binary code)

• HCI dispatcher identification
  ▪ Dispatcher candidate scanning
    o Scan for functions with the bitwise operation pattern
  ▪ Dispatcher candidate verification
    o HCI command semantics
      • Symbolically execute each candidate
      • Check whether expected value is accessed

• HCI handler identification and debloating

---

<table>
<thead>
<tr>
<th>Command</th>
<th>OCF</th>
<th>Command Parameters</th>
<th>Return Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCI_Read_BD_ADDR</td>
<td>0x0009</td>
<td></td>
<td>Status, BD_ADDR</td>
</tr>
</tbody>
</table>

---

7.4.6 Read BD_ADDR command
Implementation and Evaluation

• Implementation
  ▪ Host code analysis
    ○ LLVM pass
  ▪ Firmware analysis
    ○ angr

• Evaluation platform
  ▪ Host code
    ○ Android 6 (Bluedroid), Android 9 (Fluoride), Linux (BlueZ), Embedded system (BlueKitchen)
  ▪ Firmware
    ○ BCM4335, BCM43430A1, CYW20735B1
Host code debloating

• Keep each of the possible profiles on 4 platforms
  ▪ BlueDroid (6 profiles): average ~40% code reduction
  ▪ Fluoride (7 profiles): average ~33.7% code reduction
  ▪ BlueZ (5 profiles): average ~31.7% code reduction
  ▪ BlueKitchen (8 profiles): average ~49.1% code reduction

• Keep 5 common profile combinations
  ▪ A2DP & HFP, GATT & HFP, A2DP & GATT & HFP, A2DP & GATT, A2DP & HID
  ▪ Code reduction drops slightly (~5%) compared with keeping one profile
Firmware debloating

• BCM$_{4339}$
  ▪ ~65% of the HCI command handlers are debloated

• BCM$_{43430A1}$
  ▪ ~57% of the HCI command handlers are debloated

• CYW$_{20735B1}$
  ▪ ~83% of the HCI command handlers are debloated
Security improvement

• Prevented vulnerabilities
  ▪ 20 known vulnerabilities can be prevented by debloating different profiles
  ▪ 15 of them can be triggered over-the-air

• Prevented attacks
  ▪ BlueBorne attack (Armis’2017)
  ▪ BadBluetooth attack (NDSS’2019)
Summary

• We develop a new technique to identify the unneeded code in a Bluetooth stack with a given Bluetooth app

• We build LightBlue to automatically debloat unneeded code in Bluetooth host source code and firmware binary

• We evaluate LightBlue on 4 platforms
  ▪ 31% - 49% host code reduction and 57% - 83% firmware reduction
  ▪ Prevention of 20 known CVEs

• LightBlue is open-source
  ▪ https://github.com/purseclab/lightblue
Thank you!

Questions?
wu1220@purdue.edu

This project was supported in part by ONR under grants No0014-18-1-2674 and No0014-17-1-2513