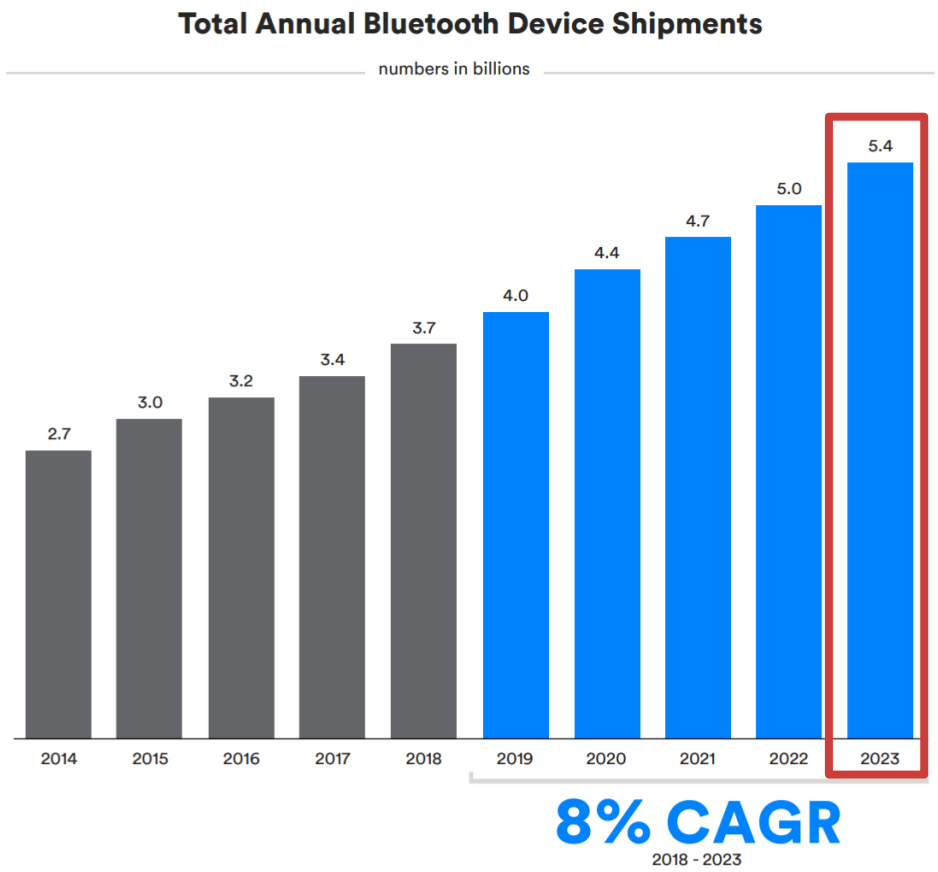
LIGHTBLUE: Automatic Profile-Aware Debloating of Bluetooth Stacks

Jianliang Wu[1], Ruoyu Wu[1], Daniele Antonioli[2], Mathias Payer[2], Nils Ole Tippenhauer[3], Dongyan Xu[1], Dave (Jing) Tian[1], Antonio Bianchi[1]

[1] Purdue University [2] EPFL [3] CISPA

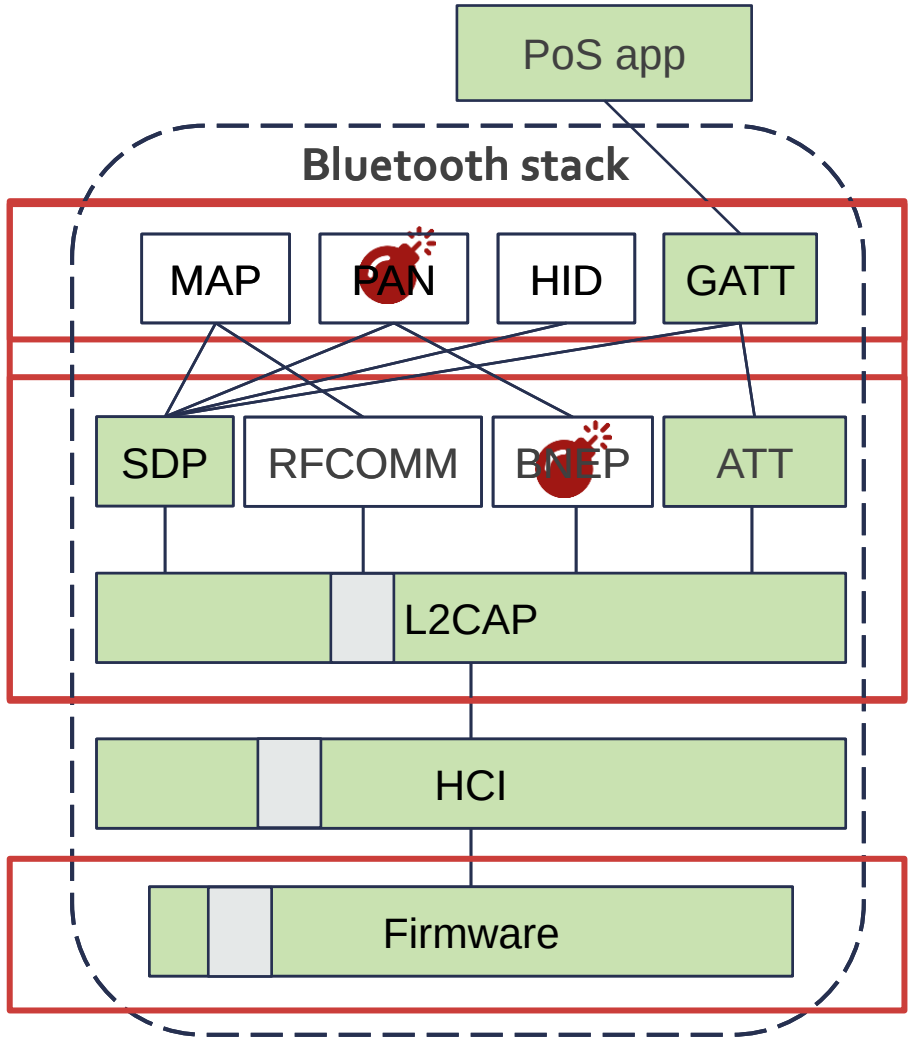
Background

- Bluetooth devices are everywhere
- Support different use cases
  - Audio streaming
  - Printing
  - Smart home
  - Health care

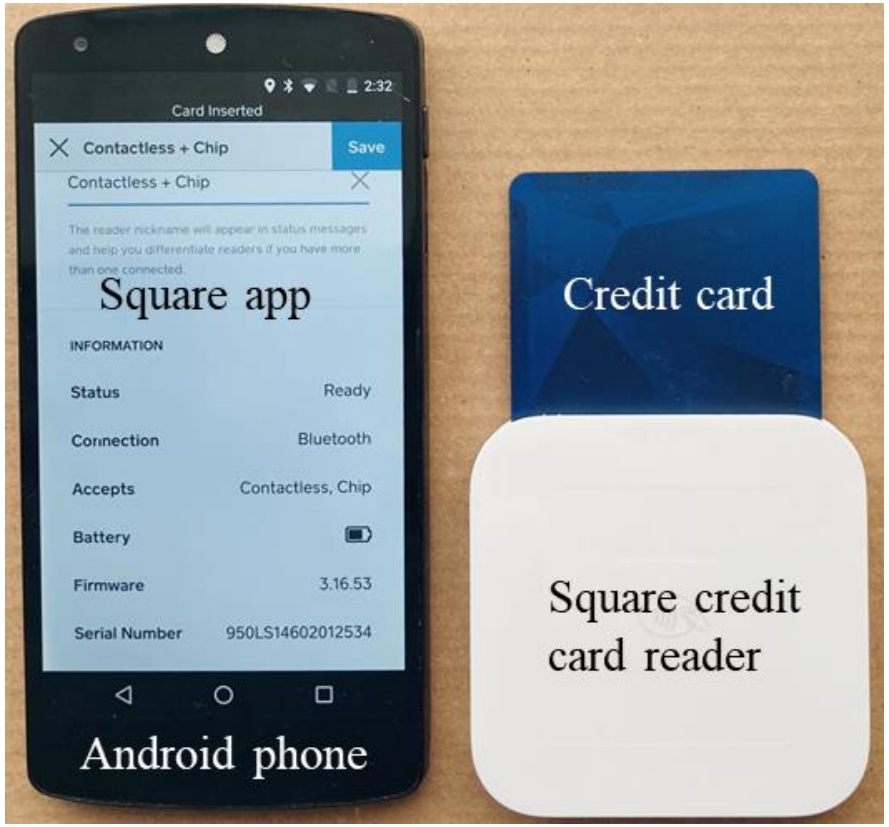


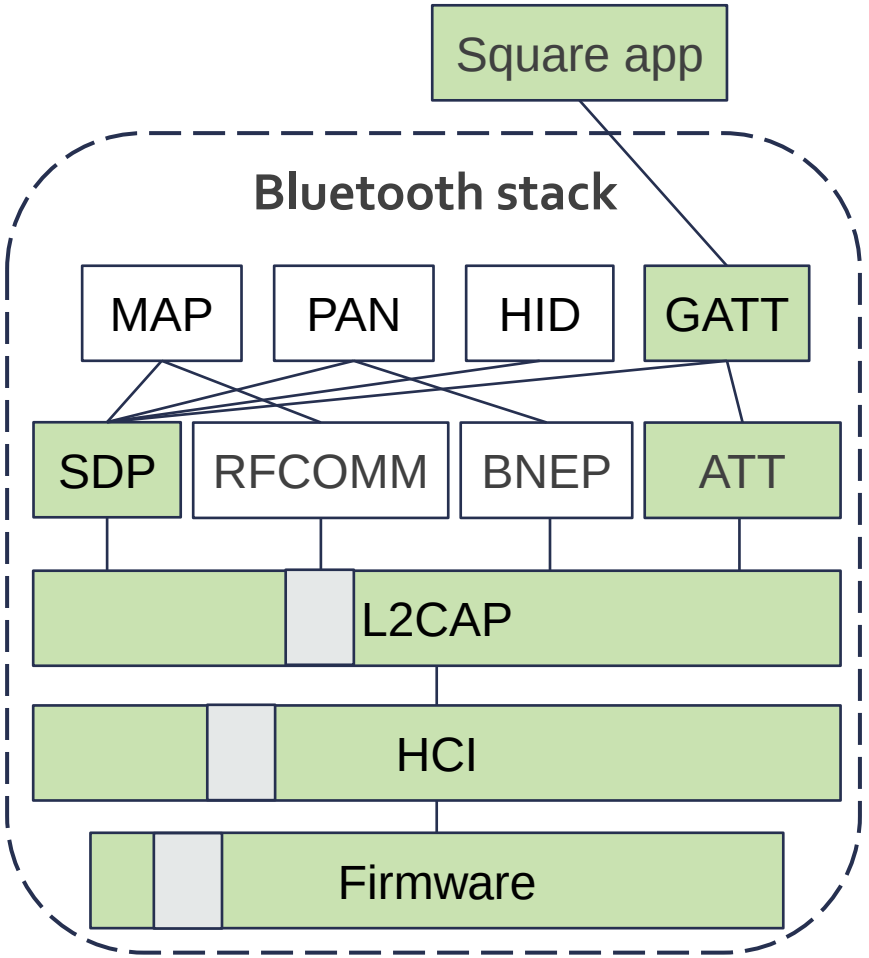
Motivation - Bluetooth is bloated

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



Motivation - Example



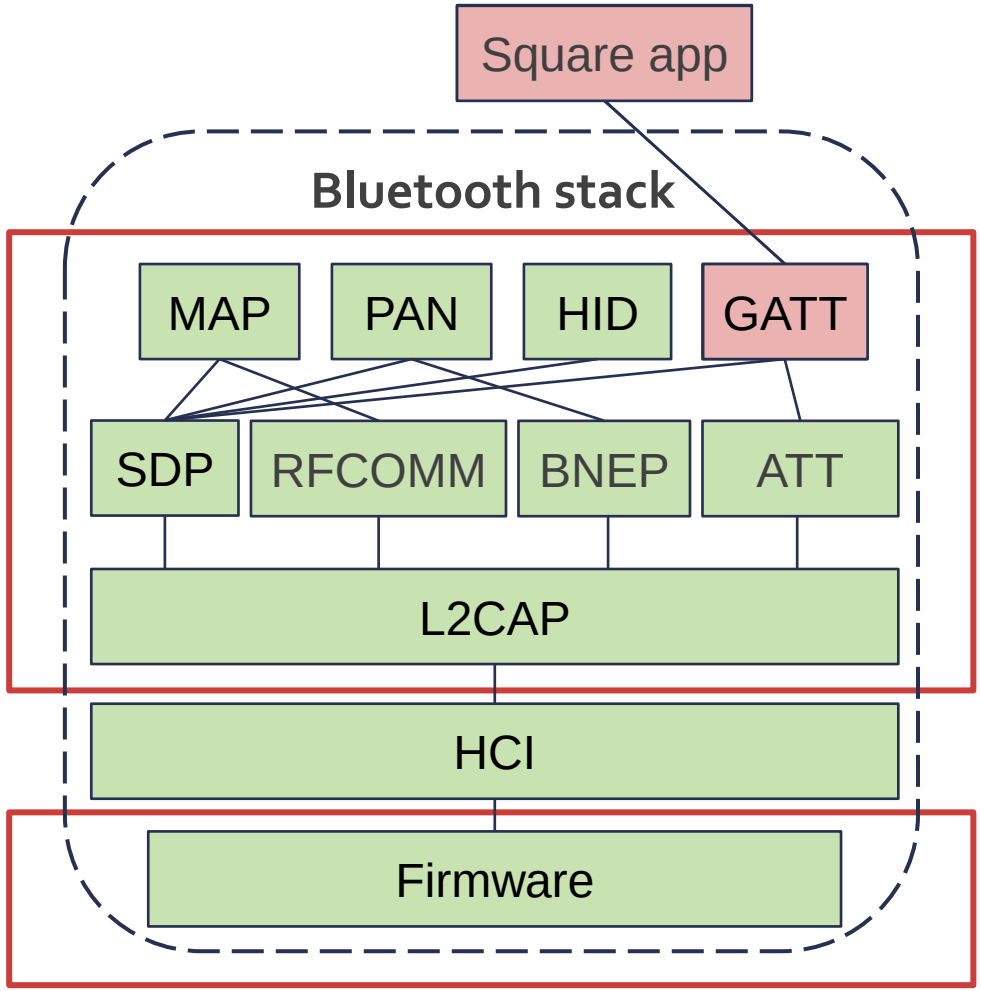


Objectives

- Debloating unneeded code so that the vulnerabilities within the unneeded code are no longer exploitable.
- Automatic
  - LightBlue should automatically removes unneeded code
- Flexible
  - LightBlue should support debloating different profiles
- Full stack debloating
  - Debloating across different components (host code and firmware)

LightBlue workflow

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

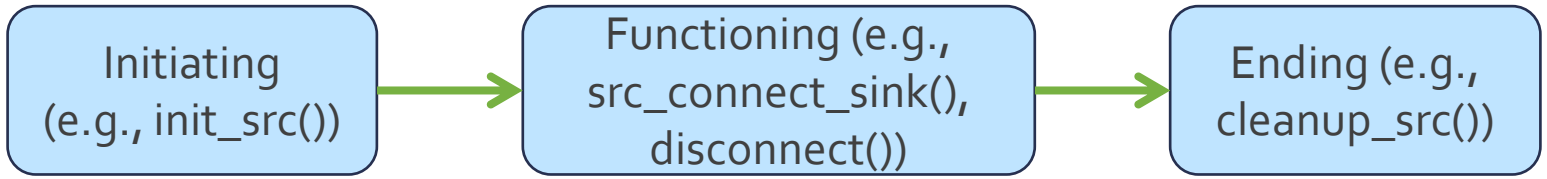
```
btif_in_execute_service_request(tBTA_SERVICE_ID service_id,
         BOOLEAN b_enable){
    switch (service_id)
    {
         case BTA_HFP_SERVICE_ID:
         case BTA_HSP_SERVICE_ID:{
            btif_hf_execute_service(b_enable);
         }break;
         case BTA_A2DP_SOURCE_SERVICE_ID:{
            btif_av_execute_service(b_enable);
         }break;
         case BTA_A2DP_SINK_SERVICE_ID:{
            btif_av_sink_execute_service(b_enable);
         }break;
         case BTA_HID_SERVICE_ID:{
             btif_hh_execute_service(b_enable);
```

```
/* A2DP profile */
bt_status_t btif_av_init(){
    if (btif_av_cb.sm_handle == NULL){
        if (!btif_a2dp_start_media_task())
            return BT_STATUS_FAIL;
        btif_av_cb.sm_handle =
        btif_sm_init((const btif_sm_handler_t*)btif_av_state_handlers,
                BTIF_AV_STATE_IDLE);
        btif_enable_service(BTA_A2DP_SOURCE_SERVICE_ID);
        btif_a2dp_on_init();
    }
```

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

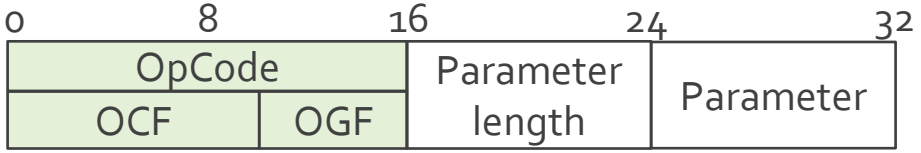
```
static const btav_interface_t bt_av_src_interface = {
    sizeof(btav_interface_t),
    init_src,
    src_connect_sink,
    disconnect,
    cleanup_src,
};
```



- Code removal and HCI commands extraction

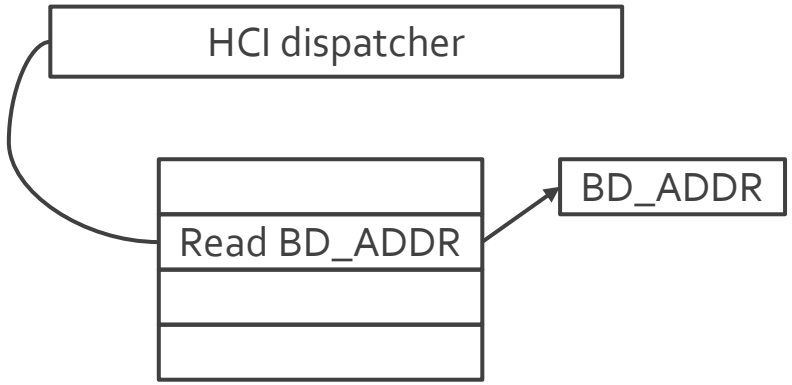
Step 3. Firmware debloating (binary code)

- HCI dispatcher identification
  - Dispatcher candidate scanning
    - Scan for functions with the bitwise operation pattern
  - Dispatcher candidate verification
    - HCI command semantics
      - Symbolically execute each candidate
      - Check whether expected value is accessed
- HCI handler identification and debloating



7.4.6 Read BD_ADDR command

Command	OCF	Command Parameters	Return Parameters
HCI_Read_BD_ADDR	0x0009		Status, BD_ADDR



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

- BCM4339
  - ~65% of the HCI command handlers are debloated
- BCM43430A1
  - ~57% of the HCI command handlers are debloated
- CYW20735B1
  - ~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)

Vul. Loc.	Related Profile	Platform	# of Vul. Functions	CVE Number
HO	A2DP & (AVRCP)	Plt. 1	4	CVE-2018-9542*
				CVE-2018-9450*
				CVE-2017-13266*
		Plt. 4	7	CVE-2018-9453
				CVE-2019-2227*
				CVE-2018-9588*
				CVE-2018-9507*
				CVE-2018-9506*
	PAN	Plt. 1	7	CVE-2019-2049
				CVE-2017-0783*
				CVE-2017-0782*
				CVE-2017-0781*
				CVE-2018-9436*
				CVE-2018-9356*
				CVE-2018-9357
				CVE-2017-13269
	MAP	Plt. 4	1	CVE-2018-9505*
	HSP/HFP	Plt. 4	5	CVE-2018-9583*
	N/A	Plt. 4	1	CVE-2019-2226
FM	GATT	Plt. 3	1	CVE-2019-13916*

*: CVEs that can be triggered by over-the-air attacks.

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 N00014-18-1-2674 and N00014-17-1-2513