Northeastern University
Systems Security Lab

Dynamic Binary Instrumentation on Android

Breakpoint 2012

Collin Mulliner, October 17-18 2012, Melbourne, Australia
crm[at]ccs.neu.edu

$ finger collin@mulline.org

- Postdoc 'Security' Researcher
  - $HOME = Northeastern University, Boston, USA
  - `cat .project`

    specialized in *mobile handset security*

- Past work
  - Some Bluetooth security work
  - A lot on SMS and MMS security
  - Mobile web usage and privacy
  - Some early work on NFC phone security

Bug Hunting on Android

- We want to find bugs!
  - fun, fame, money ;-)

- I like special targets
  - SMS (rild) done [x]
  - NFC (com.android.nfc) done [x]
    - found some simple stuff without automation
    - Makes a good example!

- **This talk will be about NFC on Android!**

Debug, monitor, simple instrumentation

- Monitor and debug
  - ADB logcat
    - Detect crashes
  - GDB
    - Debug, if you actually get it to work stable :-/
- Simple instrumentation
  - LD_LIBRARY_PRELOAD
    - Intercept or replace library calls
  - Replace library
    - Overwrite functions to intercept
    - Load original library

Dynamic Binary Instrumentation

- **Change a process at runtime**
  - **Basically: inject own code into process!**
- Debugging
  - exploit development
- Tracing and logging
  - build call graph, e.g. dynamic calls and/or callbacks
- Extract “data”
  - cryptographic keys
- Change program behavior
  - Patch checks (e.g. always return TRUE)
  - Emulation code (e.g. for fuzzing)

Dynamic Binary Instrumentation: Basics

- **Create** “instrument”
  - e.g. I/O logger
- **Inject** instrument code
  - So it can be called
- **Patch** target process
  - Make it call the instrument
- **Enjoy** the “music”

Tasks

- We need to inject code
  - Where to put code?
  - How to inject?

- Inject shared library
  - Cannot just load library from external process
  - Need to make process load the library

Inject Library (known technique!)

- ptrace() process
  - Save current state → the registers
- Write library loading code to stack; call to dlopen()
  - Including saved registers
- Call mprotect() to make stack executable
  - Write PC via ptrace(), point LR to stack
- Execute loading code → call dlopen()
  - Repair stack frame (using saved registers)
- dlopen() calls library _init() function
  - Our code executes inside the target process

Load Library

- Executes on the stack

```
// call dlopen(), cleanup stack and continue where halted
unsigned int sc[] = {
// libname
0xe59f0030, // ldr     r0, [pc, #48] / addr of "libname" in r0
0xe3a01000, // mov     r1, #0        / r1 = 0 (flags=0)
0xe1a0e00f, // mov     lr, pc        / populate lr
0xe59ff028, // ldr     pc, [pc, #40] / call dlopen()
0xe59fd01c, // ldr     sp, [pc, #28] / fix sp
0xe59f0008, // ldr     r0, [pc, #12] / fix r0
0xe59f1008, // ldr     r1, [pc, #12] / fix r1
0xe59fe008, // ldr     lr, [pc, #12] / fix lr
0xe59ff008, // ldr     pc, [pc, #12] / fix pc (continue process)
0xe1a00000, // nop (mov r0,r0)       / r0
0xe1a00000, // nop (mov r0,r0)       / r1
0xe1a00000, // nop (mov r0,r0)       / lr
0xe1a00000, // nop (mov r0,r0)       / pc
0xe1a00000, // nop (mov r0,r0)       / sp
0xe1a00000, // nop (mov r0,r0)       / addr of libname
0xe1a00000  // nop (mov r0,r0)       / dlopen address
};
```

Where is dlopen()?

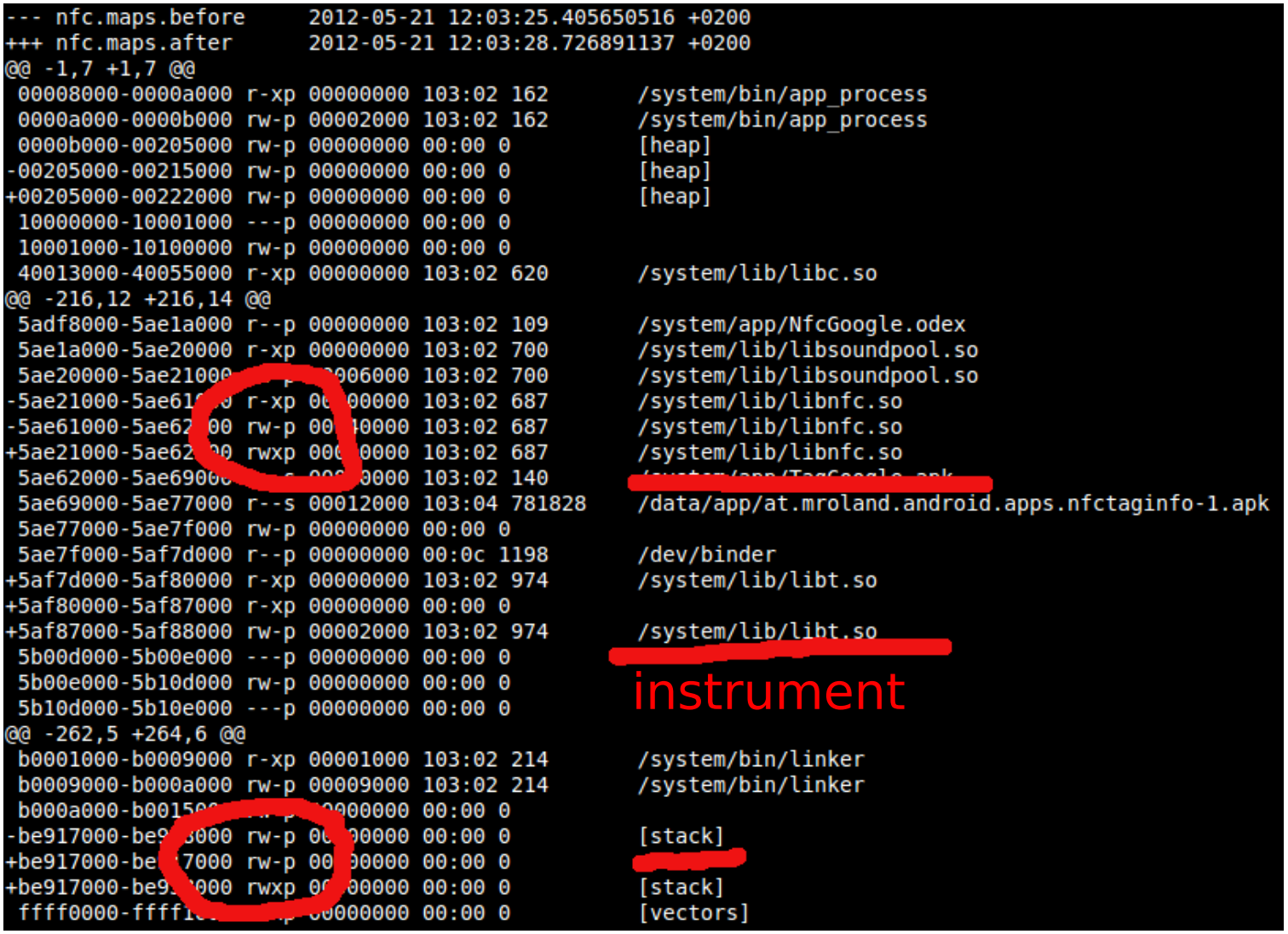
- Need to put address of dlopen() into loader code on stack
- The Android Linker knows
  - /system/bin/linker
- Find libdl_info struct
  - Find string “libdl.so”
- Get address of dlopen
  - Follow symbol table (soinfo->symtab)
- Different address on all devices I tested

Where is dlopen()?

- libdl.so is part of the linker binary (/system/bin/linker)
  - dlfcn.c (from the Android linker)

```
 * (see _elf_lookup())
 *
 * Note that adding any new symbols here requires
 * stubbing them out in libdl.
 */
static unsigned libdl_buckets[1] = { 1 };
static unsigned libdl_chains[7] = { 0, 2, 3, 4, 5, 6, 0 };

soinfo libdl_info = {
    name: "libdl.so",
    flags: [redacted]

    strtab: ANDROID_LIBDL_STRTAB,
    symtab: libdl_symtab,
    [redacted]

    nbucket: 1,
    nchain: 7,
    bucket: libdl_buckets,
    chain: libdl_chains,
};
```

Hooking com.android.nfc



My instrumentation toolkit

- Instrumentation framework aka hooking library
- Hook code stub generator
- Compile helper
  - Compiles ARM or Thumb depending on hook target
    - Possible for each individual hook
  - Deal with Android specific linking
  - Assembles the final 'instrument' library (.so file)

The Instrumentation 'Framework'

- Function address lookup
- Insert & remove hook
- Call original function

- Easy access to auxiliary data

```
struct hook_t {
    unsigned int jump[3];
    unsigned int store[3];
    unsigned char jumpt[12];
    unsigned char storet[12];
    unsigned int orig;
    unsigned int patch;
    unsigned char thumb;
    unsigned char name[128];
    void *data;
};

void hook_precall(struct hook_t *h);
void hook_postcall(struct hook_t *h);
 int hook(struct hook_t *h, int pid, char *libname, char *funcname, void *hookf);
void unhook(struct hook_t *h);
```

Symbol Lookup

- Code taken from: Victor Zandy (from hijack.c)
  - Thanks man!
- Read /proc/<PID>/maps
  - Get (code, library) base addresses
- Read library file
  - Parse ELF header
  - Lookup symbol
- Calculate absolute address
  = Symbol address + base address
- Not my code, I'm just a user here
  - Added some slight modifications

Symbol Lookup: my modifications

- Make it work for arbitrary libraries
  - Was artificial limited to libc
- Make it work with process that load a lot of libraries
  - Android...
- Make code segments writable, so we can patch
  - `mprotect(...,PROT_READ|PROT_WRITE|PROT_EXEC)`
  - Odd position, but:
    - If we lookup function address...
      ...likely that we actually patch it...
      ...so make code segment RWX

Installing Hooks

- Insert trampoline into entry point of target function
  - First save old instructions
  - ARM: Trampoline → LDR PC, [PC, #0] + address of hook
- Hook-function
  - Writes “saved” instructions back to patched function
- Issue
  - Instruction cache vs. Data cache
  - → flush instruction cache...
- in/out patching + cache flush has speed issues
  - Not noticeable

Hooking ARM Code

```
int hook(struct hook_t *h, int pid, char *libname, char *funcname, void *hookf)
{
    unsigned long int addr;
    int i;

    if (find_name(pid, funcname, libname, &addr) < 0) {
        log("can't find: %s\n", funcname)
        return 0;
    }

    log("hooking   %s = %x  hook = %x  target:", funcname, addr, hookf)
    strncpy(h->name, funcname, sizeof(h->name)-1);

    if (addr % 4 == 0) {
        log("ARM\n")
        h->thumb = 0;
        h->patch = (unsigned int)hookf;
        h->orig = addr;
        h->jump[0] = 0xe59ff000; // LDR pc, [pc, #0]
        h->jump[1] = h->patch;
        h->jump[2] = h->patch;
        for (i = 0; i < 3; i++)
            h->store[i] = ((int*)h->orig)[i];
        for (i = 0; i < 3; i++)
            ((int*)h->orig)[i] = h->jump[i];
    }
```

Hooking Thumb Code

- Some problems
  - Can't load PC with 32 bit value from relative address
    - ~~LDR pc, [pc, #0]~~
  - Need to preserve registers
    - Trampoline code needs to be clean

Hooking Thumb Code

```
else {
    h->thumb = 1;
    log("THUMB\n")
    h->patch = (unsigned int)hookf;
    h->orig = addr;
    h->jumpt[1] = 0xb4;
    h->jumpt[0] = 0x30; // push {r4,r5}
    h->jumpt[3] = 0xa5;
    h->jumpt[2] = 0x03; // add r5, pc, #12
    h->jumpt[5] = 0x68;
    h->jumpt[4] = 0x2d; // ldr r5, [r5]
    h->jumpt[7] = 0xb0;
    h->jumpt[6] = 0x02; // add sp,sp,#8
    h->jumpt[9] = 0xb4;
    h->jumpt[8] = 0x20; // push {r5}
    h->jumpt[11] = 0xb0;
    h->jumpt[10] = 0x81; // sub sp,sp,#4
    h->jumpt[13] = 0xbd;
    h->jumpt[12] = 0x20; // pop {r5, pc}
    h->jumpt[15] = 0x46;
    h->jumpt[14] = 0xaf; // nop to pad to 4 byte boundary
    memcpy(&h->jumpt[16], (unsigned char*)&h->patch, sizeof(unsigned int));
    unsigned int orig = addr - 1; // sub 1 to get real address
    for (i = 0; i < 20; i++) {
        h->storet[i] = ((unsigned char*)orig)[i];
        log("%0.2x ", h->storet[i])
    }
    log("\n")
    for (i = 0; i < 20; i++) {
        ((unsigned char*)orig)[i] = h->jumpt[i];
        log("%0.2x ", ((unsigned char*)orig)[i])
    }
```

Calling the original function

- Write back old instructions
- Flush cache

```
void hook_precall(struct hook_t *h)
{
    int i;

    if (h->thumb) {
        unsigned int orig = h->orig - 1;
        for (i = 0; i < 20; i++) {
            ((unsigned char*)orig)[i] = h->storet[i];
        }
    }
    else {
        for (i = 0; i < 3; i++)
            ((int*)h->orig)[i] = h->store[i];
    }
    my_cacheflush((unsigned int)h->orig, (unsigned int)h->orig+12);
}
```

Hook code stub generator

- Hook-Function body
  - Log when hook it is called
  - Call original function

- Hooking macro
- Auxiliary data structures

```
struct hook_t hook_phDal4Nfc_i2c_read;

struct special_phDal4Nfc_i2c_read_t {
    pphLibNfc_RspCb_t orig_cb;
    pphLibNfc_RspCb_t my_cb;
} special_phDal4Nfc_i2c_read;

#define HOOK_phDal4Nfc_i2c_read \
hook(&hook_phDal4Nfc_i2c_read, pid, "libnfc", "phDal4Nfc_i2c_read", my_phDal4Nfc_i2c_read);\
hook_phDal4Nfc_i2c_read.data = &special_phDal4Nfc_i2c_read;\
memset((char*)&special_phDal4Nfc_i2c_read, 0, sizeof(special_phDal4Nfc_i2c_read));
```

Developing an Instrument

- Identify the functions you want to hook
  - Reverse engineer binary, read source, … your task!
- Pitfalls when developing your instrument
  - Make sure lib functions are available in target process
    - Otherwise library does not load!
  - Log to a file, stdout/stderr not available
    - /data/local/tmp is the place

Developing an Instrument cont.

- Where to put library
  - Put it in → /system/lib
    - Requires: # mount -o remount,rw /system

- I want to use dl___() in my code but it hangs!
  - Don't call dl___() in your library's _init() function
  - Use my symbol lookup code in _init()
  - Call dl___() from:
    - A thread
    - A patched function

My Instruments for NFC

- Log I2C
  - Sniff com between NFC stack process and NFC chip
  - Nexus S
  - actually contributed by Charlie!
- Log Uart
  - Sniff com between NFC stack process and NFC chip
  - Galaxy Nexus
- Sniff
  - Log NDEF read (dump NDEF payload)
- EmuNFCcard
  - Software emulate reading an NFC card (**for fuzzing!**)

Simple “i2c sniffing” hooking code

```
void my_init()
{
    log("libt loaded...\n")
    // required by macros
    int pid = getpid();

    HOOK_phDal4Nfc_i2c_read
    HOOK_phDal4Nfc_i2c_write
}
```

A hook in action: i2c_read

- Get hook struct
  - Extract: original function address & data pointer
- Call original function
  - Remove hook, call function, insert hook
- Dump data

```
int my_phDal4Nfc_i2c_read(uint8_t *pBuffer, int nNbBytesToRead)
{
    orig_phDal4Nfc_i2c_read = (void*) hook_phDal4Nfc_i2c_read.orig;
    int i;
    struct special_phDal4Nfc_i2c_read_t *d = (struct special_phDal4Nfc_i2c_read_t*)hook_phDal4Nfc_i2c_read.data;

    hook_precall(&hook_phDal4Nfc_i2c_read);
    NFCSTATUS res = orig_phDal4Nfc_i2c_read(pBuffer, nNbBytesToRead);
    hook_postcall(&hook_phDal4Nfc_i2c_read);

    log("--read %d bytes --\n", nNbBytesToRead)
    for (i = 0; i < nNbBytesToRead; i++) {
        log("%0.2x", pBuffer[i])
    }
    log("\n")

    log("%s result = %x\n", __func__, res)
    return res;
}
```

I2C sniff output

```
libt loaded...
phDal4Nfc_i2c_read = 0x5b1ab2e8 hook = 0x57926f8c target:ARM
phDal4Nfc_i2c_write = 0x5b1ab0ac hook = 0x57927124 target:ARM
--read 16 bytes --
581805cb4d00000000000000000000000
my_phDal4Nfc_i2c_read result = 5
--read 1 bytes --
06
my_phDal4Nfc_i2c_read result = 1
--write 7 bytes --
06a18502029a6f
my_phDal4Nfc_i2c_write result = 7
--write 4 bytes --
03c1aaf2
my_phDal4Nfc_i2c_write result = 4
--read 12 bytes --
8d858004b8b1f24b28808816
my_phDal4Nfc_i2c_read result = c
--write 7 bytes --
06aa85020306be
my_phDal4Nfc_i2c_write result = 7
--read 6 bytes --
9685800027e8
my_phDal4Nfc_i2c_read result = 6
```

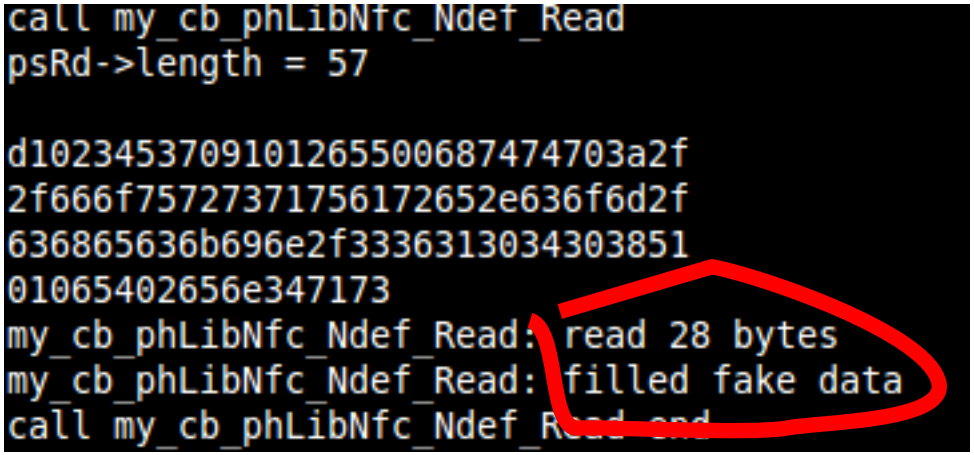
RFID/NFC Card Read Sniff Payload

- hook: phLibNfc_Ndef_Read(...)
  - Completely asynchronous
  - Ndef_Read(..) takes pointer to callback
  - Callback indicates data read
    → **patch callback to get data**

```
my_phLibNfc_Ndef_Read enter
orig_phLibNfc_Ndef_Read = 5b17aa38
my_phLibNfc_Ndef_Read result = d
call my_cb_phLibNfc_Ndef_Read
psRd->length = 55

d1023253709101145500687474703a2f
2f7777772e68656973652e6465510116
5402656e687474703a2f2f736c617368
646f742e636f6d
call my_cb_phLibNfc_Ndef_Read end
```

RFID/NFC Card Read ~~Sniff~~ Replace Payload

- hook: phLibNfc_Ndef_Read(...)
  - Completely asynchronous
  - Ndef_Read(..) takes pointer to callback
  - Callback indicates data read
    → **patch callback to replace data**



How do we fuzz tag reading?

- We can replace data read from tag
  - Don't need to write “fuzz” payload to tag
  - Just read same tag over and over but replace payload
    - Improves NFC fuzzing speed from 2008

How do we fuzz tag reading?

- We can replace data read from tag
  - Don't need to write “fuzz” payload to tag
  - Just read same tag over and over but replace payload
    - Improves NFC fuzzing speed from 2008

- But this is still lame
  - Want full automation, without touching the phone!

Finally: fully automated RFID/NFC tag fuzzing

- Idea
  - Simulate a card being read by the NFC chip
    → data pushed up the NFC stack for parsing
- Fuzz com.android.nfc
  - generate NDEF tag content and inject into process
- Result
  - NFC tag fuzzing without need to write data to tag
    → no need to hold tag to the phone

Fuzzing, Networking, and Android Permissions

- Target process might **not have** network permissions
  - e.g. our target com.google.nfc
- Fuzzing requires getting “data” to the phone
  - ...to the fuzzed process
- Simple solution
  - Use file system, put “fuzz data” in file and read it

Fuzzing, Networking, and Android Permissions

- Target process might **not have** network permissions
  - e.g. our target com.google.nfc
- Fuzzing requires getting “data” to the phone
  - ...to the fuzzed process
- Simple solution
  - Use file system, put “fuzz data” in file and read it
- **Dude laaaame! We want “network”**
  ...mmmh okay...

Network “Emulation” aka a file descriptor

- A file descriptor to
  - read(), write(), poll(), select()

- What about a pseudo terminal?

```
void start_coms()
{
    // workaround for missing socket permission :)
    coms = open("/dev/ptmx", O_RDWR|O_NOCTTY);
    if (coms <= 0)
        log("posix_openpt failed\n")
    else
        log("pt ok\n")
    if (unlockpt(coms) < 0)
        log("unlockpt failed\n")
    log("pty name: %s\n", ptsname(coms))

    struct termios  ios;
    tcgetattr(coms, &ios);
    ios.c_lflag = 0;   /* disable ECHO, ICANON, etc... */
    tcsetattr(coms, TCSANOW, &ios);
}
```

Network “Emulation” finalized via proxy

- Simple proxy tool that ...

```
for (;;)
    bind(), listen(), accept()
    open(pts)
    read(net)
    write(pts)
```

- Now target binary “has” network
  - We can delivery “fuzz data”
    (tested on Android 2.3 and 4.0.4)

Fully automated RFID/NFC tag fuzzing

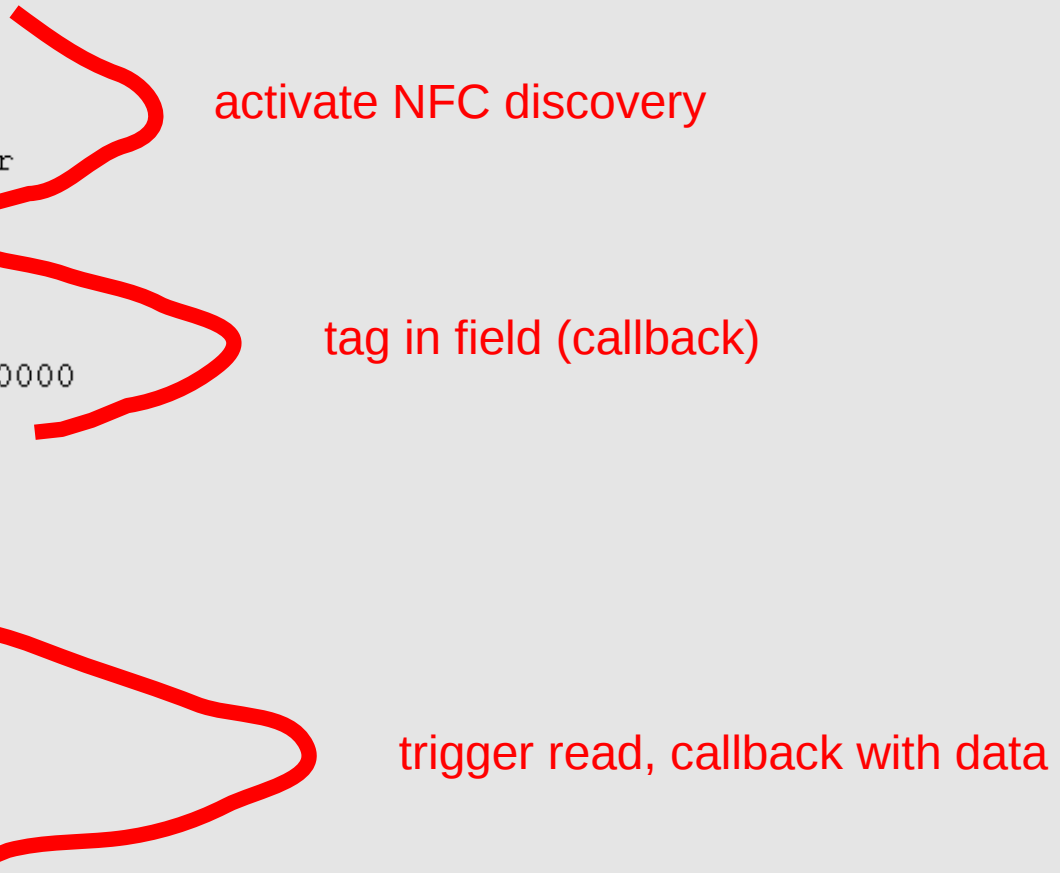
- Idea
  - Simulate a card being read by the NFC chip
    → data pushed up the NFC stack for parsing
- Fuzz com.android.nfc
  - generate NDEF tag content and inject into process
- Result
  - NFC tag fuzzing without need to write data to tag
    → no need to hold tag to the phone

Inside com.android.nfc

- Spawned by app_process (zygote)
- Two main libraries
  - libnfc.so and libnfc_jni.so ← native interface
- libnfc → libnfc-nxp
  - Completely asynchronous operation
    - Callback indicate end of operation
- libnfc_jni
  - Calls libnfc functions, provides callback functions
  - runs extra thread for processing libnfc's message queue

Tag Detect-Read call stack: com.android.nfc

```
phLibNfc_Mgt_ConfigureDiscovery
 cb_phLibNfc_Mgt_ConfigureDiscovery

phLibNfc_RemoteDev_NtfRegister

phLibNfc_Mgt_ConfigureDiscovery enter
 cb_phLibNfc_Mgt_ConfigureDiscovery

cb_phLibNfc_RemoteDev_NtfRegister
 uNofRemoteDev = 1
 Status = 0
 remdevtype c
 uuid length = 4 id=da60c7130000000000000

phLibNfc_RemoteDev_Connect

phLibNfc_RemoteDev_Connect

phLibNfc_Ndef_CheckNdef

phLibNfc_Ndef_Read
 cb_phLibNfc_Ndef_Read
  psRd->length = 55

   d10232537091011455006874747 03a2f
   2f7777772e68656973652e6465510116
   5402656e687474703a2f2f736c617368
   646f742e636f6d

phLibNfc_RemoteDev_Connect

phLibNfc_RemoteDev_CheckPresence
 cb_phLibNfc_RemoteDev_CheckPresence

phLibNfc_RemoteDev_CheckPresence
 cb_phLibNfc_RemoteDev_CheckPresence
```



Full NFC/RFID (NDEF) tag read emulation

- Network communication using our pts proxy technique
  - Handled by a thread started in _init() of instrument

Full NFC/RFID (NDEF) tag read emulation

- Network communication using our pts proxy technique
  - Handled by a thread started in _init() of instrument
- Some obstacles
  - Cannot call callbacks from our thread
    - Results in just a crash
  - Need to call callbacks from libnfc_jni's libnfc-thread
    - How???

Full NFC/RFID (NDEF) tag read emulation

- Network communication using our pts proxy technique
  - Handled by a thread started in _init() of instrument
- Some obstacles
  - Cannot call callbacks from our thread
    - Results in just a crash
  - Need to call callbacks from libnfc_jni's libnfc-thread
    - How???
- Libnfc has a internal messaging system (phDal4Nfc_msg*)
  - _msgrecv() called in libnfc_jni
    - hook it and use it to issue our fake callbacks

Full NFC/RFID (NDEF) tag read emulation

- Basic idea: call “new tag” callback (registered by NtfRegister)
- Patch all intermediate calls to return SUCCESS
  - _Connect, _CheckPresence, and _CheckNdef
- Provide fake tag data to callback of Ndef_Read

```
libt loaded...
hooking   phLibNfc_Ndef_Read = 8050bc6c  hook = 807056a4  target:ARM
hooking   phLibNfc_RemoteDev_CheckPresence = 80508050  hook = 80705c80  target:ARM
hooking   phLibNfc_RemoteDev_NtfRegister = 80509328  hook = 80706904  target:ARM
hooking   phLibNfc_RemoteDev_Connect = 80508e44  hook = 80705eb8  target:ARM
hooking   phLibNfc_Ndef_CheckNdef = 8050c904  hook = 80705338  target:ARM
hooking   phLibNfc_Mgt_ConfigureDiscovery = 8050834c  hook = 80705120  target:ARM
hooking   phDal4Nfc_msgrcv = 80543698  hook = 8070495c  target:ARM
msgsend = 80543718
pt ok
pty name: /dev/pts/1
libt init done.
thread start
ifc_thread sleeping... 0
```

Full NFC/RFID (NDEF) tag read emulation

Release Android DBI Toolkit v0.2

- Breakpoint special!!!
- Contains fixed and improvements
- How includes my NFC card emulator code!
- http://www.mulliner.org/android/feed/collin_android_dbi_v02.zip

Improvements

- Real support for Thumb/Thumb2
  - Thumb is supported (not well tested)
- Remove requirement for in/out patching
  - Disassemble → assemble on-the-fly
    - Faster execution
  - GOT patching
- Support hooking at arbitrary address
  - Right now only hooking the function entry is supported

Conclusions

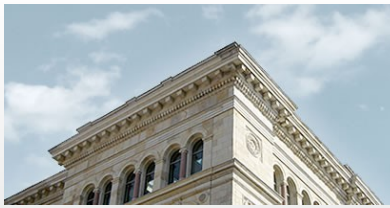
- Binary instrumentation on Android
  - Works like on other OSes
  - Need to deal with Android issues
- Now I just need to fuzz Android NFC :-)
  - Get the emulation more stable
  - Find some time for actual fuzzing
- Thanks
  - Nico
    - Good hints in the early state of this project
  - Charlie
    - For testing my framework! \o/

Related, Previous, and Similar Work

- **Dynamic Binary instrumentation is not new!**

- Android / ARM
  - Georg Wicherski
    - Thumb2 instrumentation stuff shown at HES2012
    - No details and/or code published
  - Sebastian Krahmer
    - ported his injectso tool to Android
      - Just learned this a week ago :-(

- Cydia's substrate (iOS)



EOF

Thank you! Any Questions ?

twitter: @collinrm
crm[at]ccs.neu.edu
http://mulliner.org/android/