DEBUGGING HTC PHONES BOOTLOADERS

HBOOTDBG

22/10/2013 - HACK.LU 2013

Cédric Halbronn, Nicolas Hureau

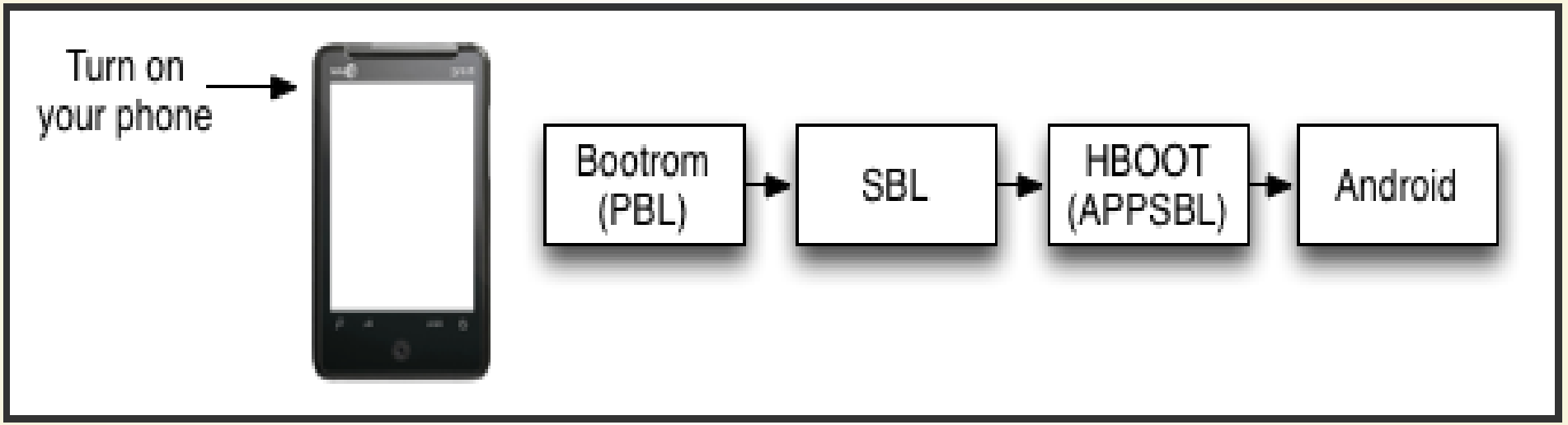
WHO ARE WE?

- Cédric Halbronn - @saidelike
  - 4 years at Sogeti ESEC Lab
  - Worked on Windows Mobile, iPhone, Android security
  - Focusing on vulnerability research & exploitation

- Nicolas Hureau - @kalenz
  - New recrue at Sogeti ESEC Lab
  - Likes low-level stuff

WHAT IS A BOOTLOADER?

- Piece of code first executed when turning on your phone



BOOTLOADER GOAL

- Initializing hardware
- Loading Google operating system (Android)
- Restore device factory state (if Android gets corrupted)
- Update the phone

REASONS TO LOOK INTO BOOTLOADERS?

- Unlocking the bootloader and rooting your device
  - Permanent root of your device
  - Install custom ROM (eg: Cyanogenmod)

- Understanding how bootloaders really work

- Very old code, good potential for vulnerabilities
- Evaluating the physical security risks
  - What does an attacker get access to?

ABOUT THIS TALK

- Debugging HTC phones bootloader

AGENDA

1. **Basics**
2. **Revolutionary vulnerability**
3. **HBOOT debugger**
4. **Simple bug**
5. **Conclusion**

AGENDA

1. Basics
2. Revolutionary vulnerability
3. HBOOT debugger
4. Simple bug
5. Conclusion

WHAT IS HBOOT?

- The **bootloader** of HTC Android phones
- Used on all HTC phones
  - Desire, Desire S, Desire Z, One, etc.
- Controlled by HTC
- Different branded Android phone ⇒ different bootloader
  (eg: Samsung, Motorola, etc.)

GETTING TO KNOW HBOOT

- Closed sources
  ⇒ HTC code base, not Android

- 2 modes: HBOOT/FASTBOOT

- Helpful references
  - **xda-developers.com**
  - **tjworld.net**
  - **unrevoked hboot-tools**

VULNERABILITIES IN HBOOT?

- Used in unlocking tools
  - **unrevoked3** (deprecated)
  - **AlphaRev** (deprecated)
  - **revolutionary**: 15 HTC devices supported
  - **Unlimited.IO**: ~10 other HTC devices supported
  - **rumrunner**: HTC One

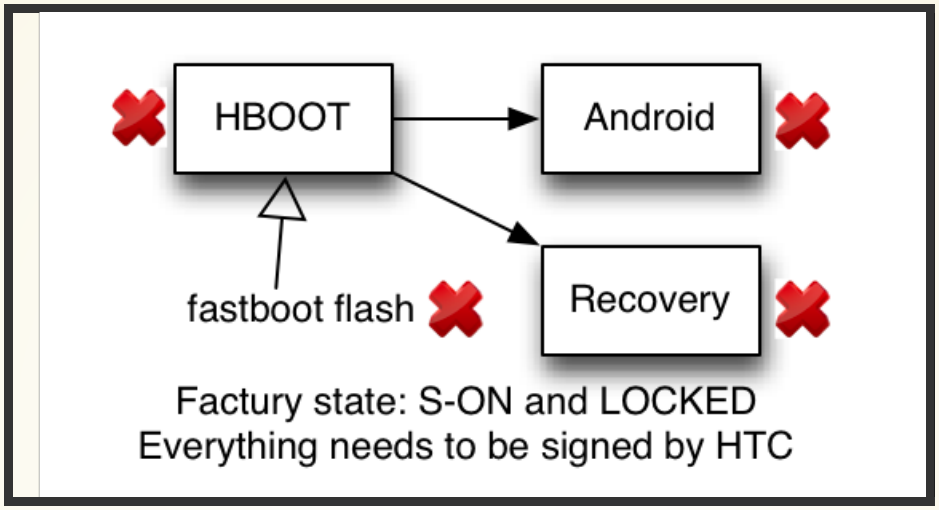
- **read_emmc** HBOOT command to read flash memory
  - HTC Desire Z (only?)

- XTC clip to S-OFF the device

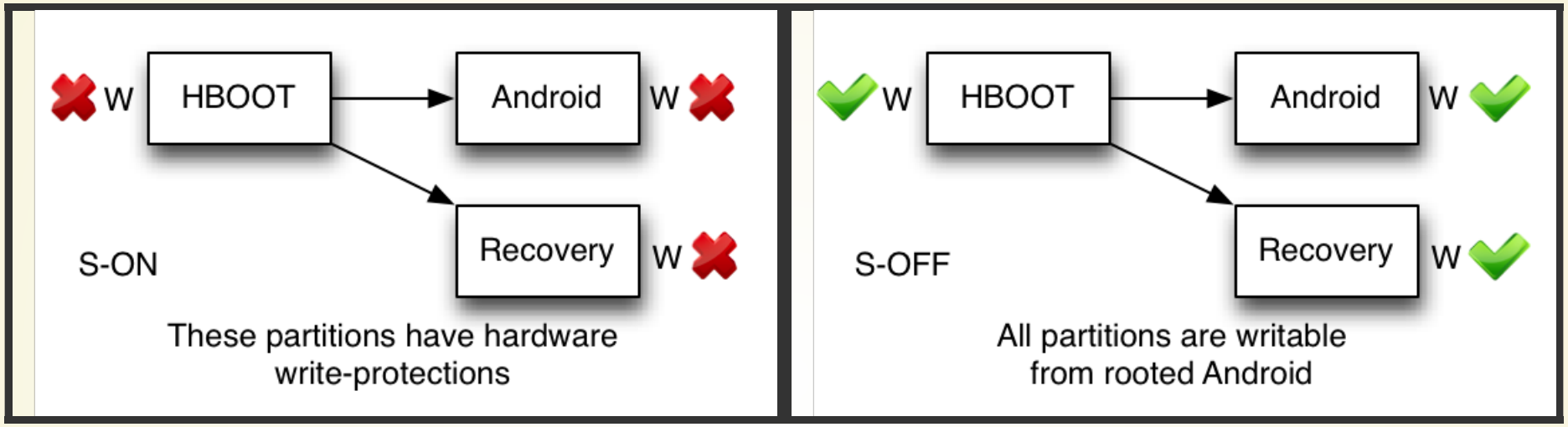
TARGETED DEVICE

- HTC Desire Z
- Run on a Qualcomm MSM7230 (Snapdragon S2) SoC
  - Baseband processor: ARM9
  - Application processor: Scorpion (custom ARMv7 design)
- Release date: 2010
- HBOOT version: 0.85

HTC SECURITY MODEL



S-ON

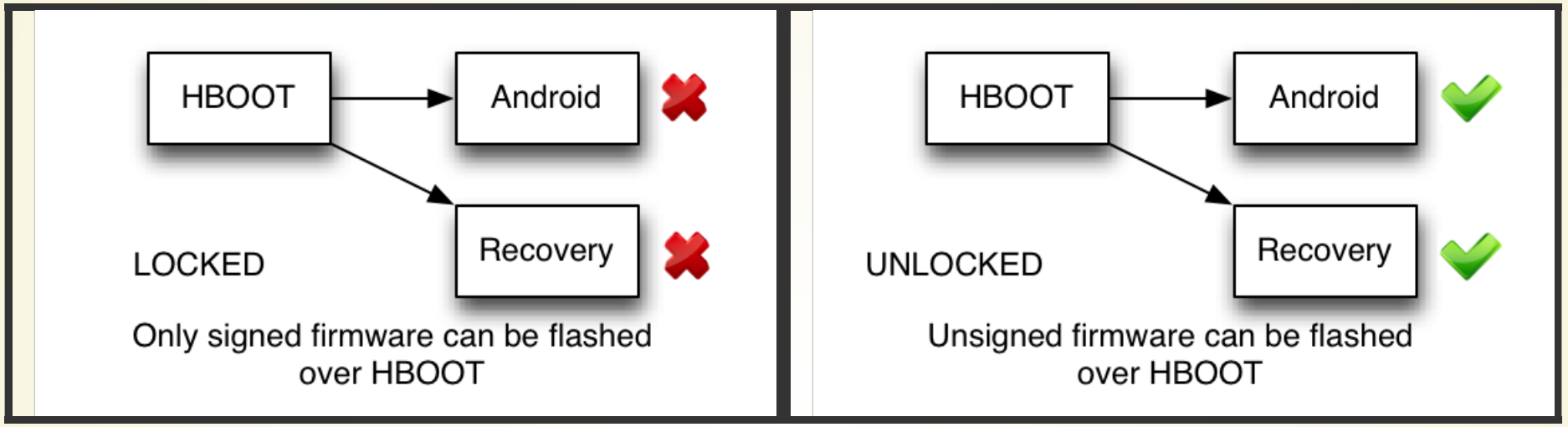

W
HBOOT
Android
W
Recovery
W
S-ON
These partitions have hardware write-protections
W
HBOOT
Android
W
Recovery
W
S-OFF
All partitions are writable from rooted Android

HTC SECURITY FLAG

- Everything must be signed by HTC
- HBOOT does not allow to flash unsigned Android ROM (zip)
- HBOOT does not allow to run unsigned code (NBH file)
- HBOOT write-protects *system* / *hboot* partitions during boot
  - It is hardware-locked (S-ON flag)

⇒ Even a root vulnerability does NOT allow to write partitions

LOCKED


HBOOT
Android
Recovery
LOCKED
Only signed firmware can be flashed over HBOOT
HBOOT
Android
Recovery
UNLOCKED
Unsigned firmware can be flashed over HBOOT

HTC LOCK/UNLOCK

- HTC allows us to unlock our device (htcdev.com)
- Unlock allows HBOOT to flash an **unsigned** *system* partition
  - HTC keeps control on HBOOT (we keep S-ON)

- From a security perspective, unlock forces a factory reset
  - Attacker can not access your data (*wipe*) (theorically)

- BUT after unlocking your device
  ⇒ Attacker could make HBOOT load unsigned code and potentially access your data

GETTING HBOOT BINARY

- HTC proprietary code
- Windows update package
  - RUU.exe contains a rom.zip file. Content of the rom.zip file

```
boot.img:              Android kernel
    hboot_XYZ.nb0:         HBOOT bootloader <- what we are looking for
    radio.img:             Baseband code
    recovery.img:          Recovery kernel
    system.img:            System partition
    userdata.img:          Data partition
```

- Static analysis (IDA Pro). Raw ARM code

DUMPING HBOOT IN RAM

- IDA not following some code paths
  - Because of uninitialized memory structures

- Initialized context ⇒ get more info on how it really works
- Need to get code execution to read memory snapshot

GETTING CODE EXECUTION IN HBOOT

- Unlock ⇒ flash custom Android
  - Not possible to load unsigned code

- S-OFF the device with XTC clip + load unsigned NBH binary?
  - Would be after HBOOT execution

- Exploit a vulnerability in HBOOT?
  - Unlock exploits = good candidates to analyze
    ⇒ Revolutionary tool

AGENDA

1. Basics
2. Revolutionary vulnerability
3. HBOOT debugger
4. Simple bug
5. Conclusion

REVOLUTIONARY

- 15 supported HTC devices
- HTC Desire Z not officially supported
  - But HBOOT still vulnerable
- Analyzed version: 0.4pre4

INTERNAL STEPS

1. Temporary "root" of the phone (*zergRush*)
2. Rewrite "misc" partition from Android
3. Reboot phone in HBOOT
   - "fastboot getvar:mainver" ⇒ flash patched HBOOT

'FASTBOOT GETVAR' HANDLER

- fastboot getvar:mainver

```
void fastboot_getvar(char* var)
    {
      char buf[64]; //stack-based buffer
      fastboot_getvar_handler(var, buf);
      usb_send(buf)
    }

    void fastboot_getvar_handler(char* var, char* buf)
    {
      if (!strcmp(var, "mainver"))
      {
        //get main version from "misc" partition
        sprintf(buf, "%s", fastboot_getvar_mainver()));
      } else {
        //...
    }
```

'FASTBOOT GETVAR' HANDLER

- "misc" partition writable from rooted Android
  - Possible to rewrite the main version
- After reboot in HBOOT
  - Stack-based buffer overflow ⇒ code execution

GETTING CODE EXECUTION IN HBOOT (CONTINUE)

- Coming back to what interests us
  - Dump HBOOT memory

- Send code implementing read/write memory primitives
  - Using regular "fastboot download" command
- Trigger revolutionary exploit to get code execution
  ⇒ Dump whole memory to have HBOOT memory context

WHAT ABOUT DEBUGGING?

- Static analysis ⇒ take time
- Would be helpful to have dynamic analysis tools
- Would look at specific behaviors
  - Command parsing, package update, Android loading, etc.

- Requirements
  - Get code execution: OK
  - Communication between phone and computer: TODO

COMMUNICATION

- HBOOT/FASTBOOT exposes a serial console over USB
- Several commands
  - Interesting ones

```
getvar <variable>                  display a bootloader variable
download [len:hexbinary]           send data to the download area
oem                                custom manufacturer commands
```

  - "download" not implemented in fastboot computer binary
- Hook one of these commands
  - fastboot oem

AGENDA

1. Basics
2. Revolutionary vulnerability
3. HBOOT debugger
4. Simple bug
5. Conclusion

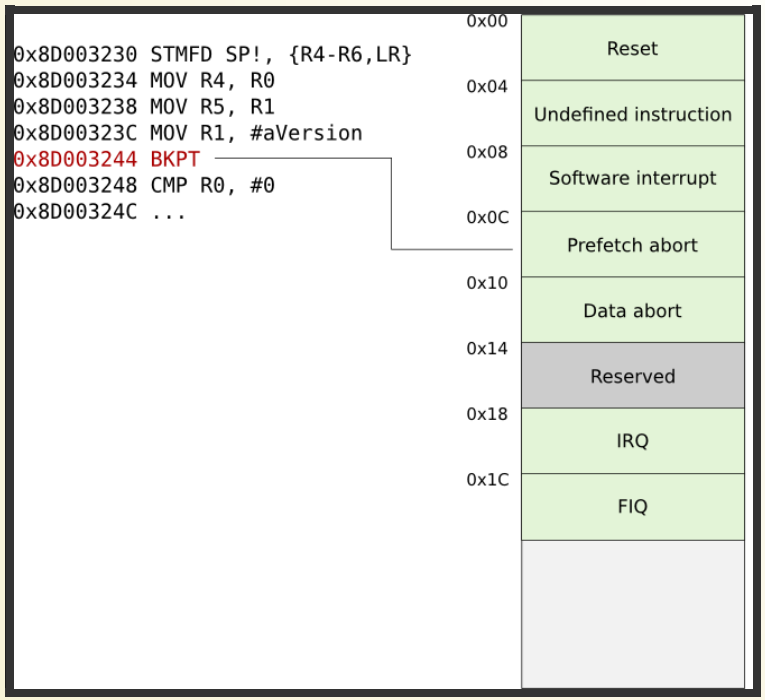
DEBUGGER

- Code execution in HBOOT + communication: OK
  ⇒ debugger implementation

- Requirements
  - Read/write memory: OK (code execution)
  - Breakpoints: TODO

BREAKPOINT IN ARM

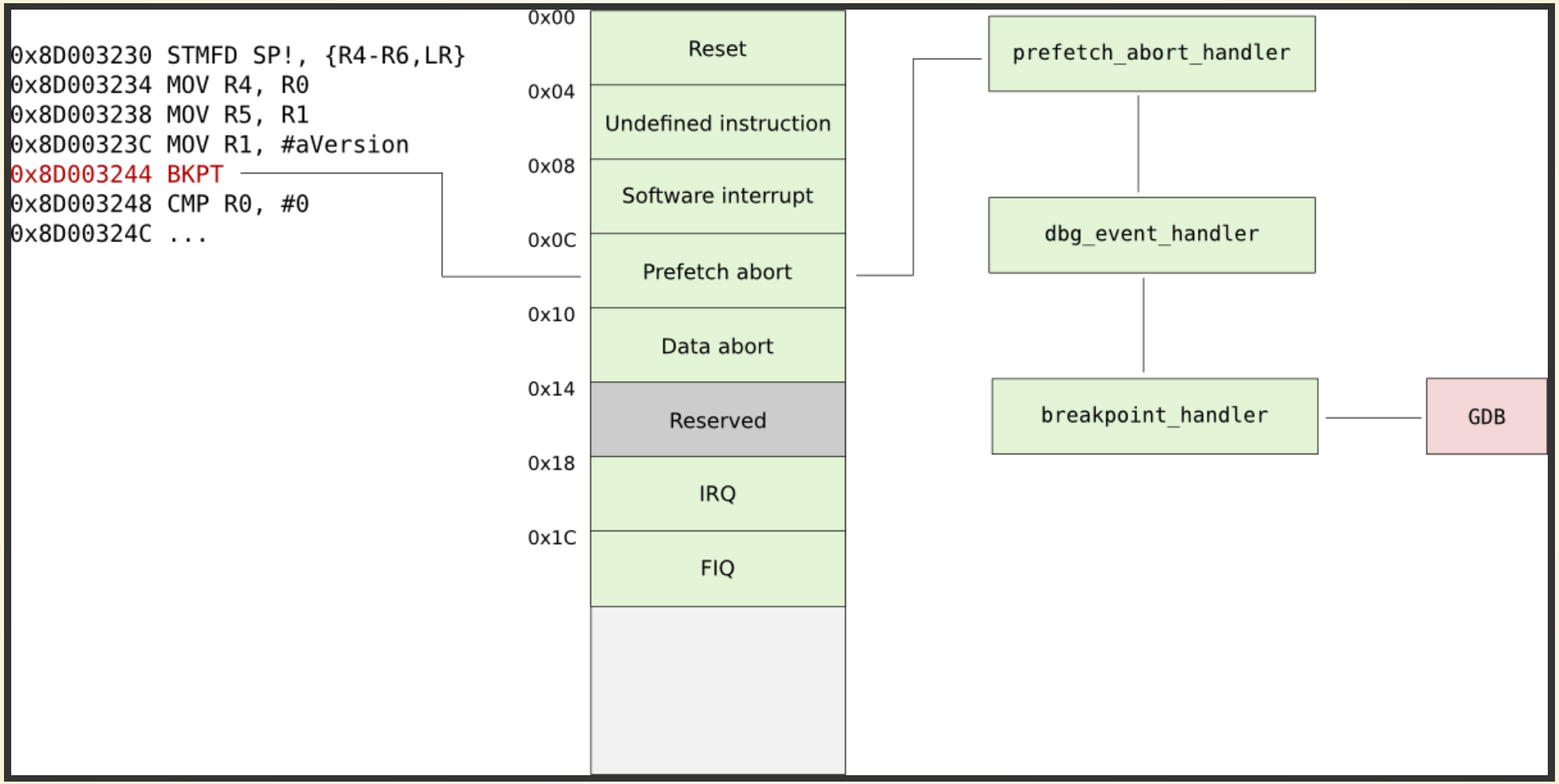
- ARM "bkpt" instruction
- When hitting a breakpoint
  - CPU triggers an exception: sets DBGDSCR.MOE to "BKPT instruction debug event"
  - Branch at offset 0xC (prefetch abort)



BREAKPOINT HANDLING IN HBOOT

- By default, no exception vector table in HBOOT
  - Install our own handler: no need to check DBGDSCR.MOE
  - Setup abort stack
- Save context (registers) to restore them after handling

BREAKPOINT HANDLING IN HBOOT



DEBUGGER

- Debugger on the phone: OK ⇒ need a debugger client
- Requirements
  - Read/write memory: OK (code execution)
  - Breakpoints: OK (hook prefetch abort)
  - Debugger client: TODO

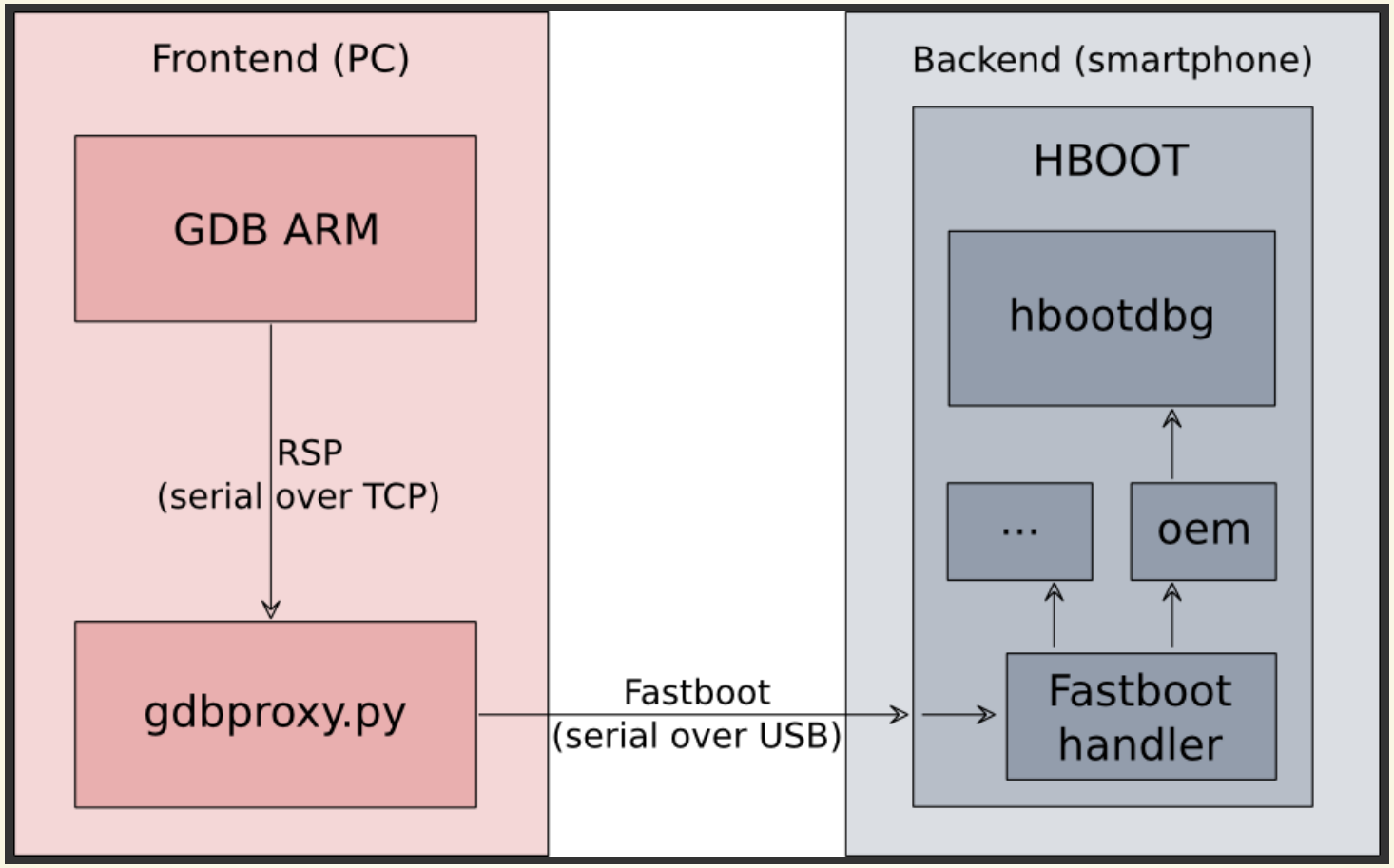
GDBPROXY.PY

- Script interfacing GDB and debugger in HBOOT
  - Works as a GDB server (RSP protocol)
  - And a client for the debugger
- Any GDB client applies: arm-gdb, IDA Pro, etc.

DEBUGGER

- Requirements
  - Read/write memory: OK (code execution)
  - Breakpoints: OK (hook prefetch abort)
  - Debugger client: OK (any gdb client)

DEBUGGER ARCHITECTURE



- Target similarities: design inspired by qcombbdbg

SUMMARY

- Revolutionary exploit to inject code (*fastboot getvar:mainver)*
- Communication with debugger (hook *fastboot oem)*
- Frontend
  - Python script proxying requests from GDB to backend
    - Handle GDB RSP and our debugger protocol
  - Read/write memory & registers
  - Add/delete breakpoints
- Backend: injected code
  - Hook exception vector: prefetch abort
    - Called when BKPT instruction decoded
  - Simple software breakpoints

WHAT ABOUT USING OUR DEBUGGER?

- Basic debugger implementation: OK
- Using our debugger: TODO

AGENDA

1. Basics
2. Revolutionary vulnerability
3. HBOOT debugger
4. Simple bug
5. Conclusion

FINDING A NON HARMFUL BUG

- In HBOOT mode ⇒ **hboot>** prompt
  - hboot> ⇔ "fastboot oem"
  - Execute commands

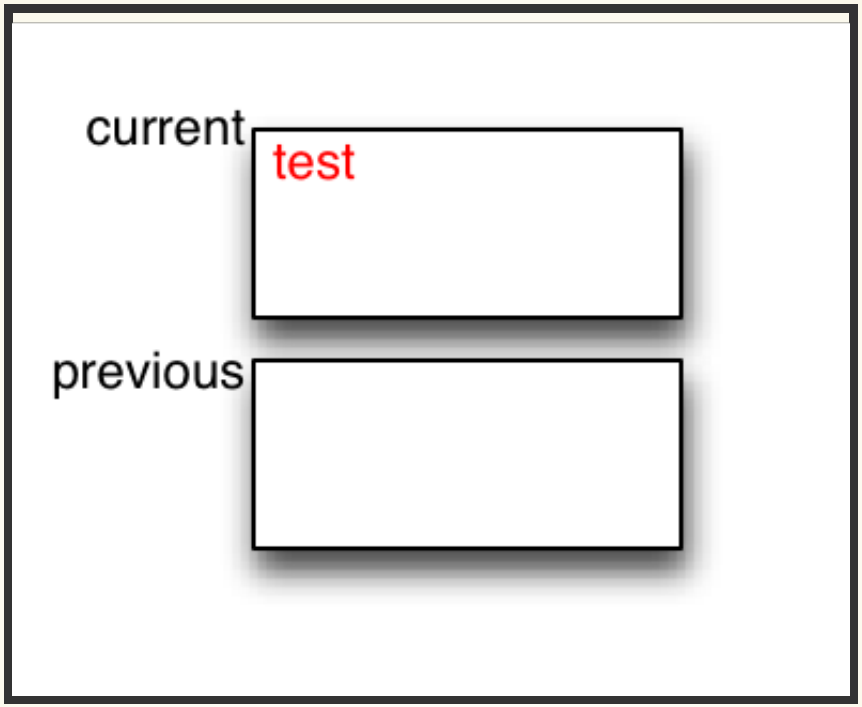
- Enter the following 2 commands
  - 'A'*256 + \n + 'B'*256 + \t\n
  - Phone not responding anymore

- How are commands parsed?

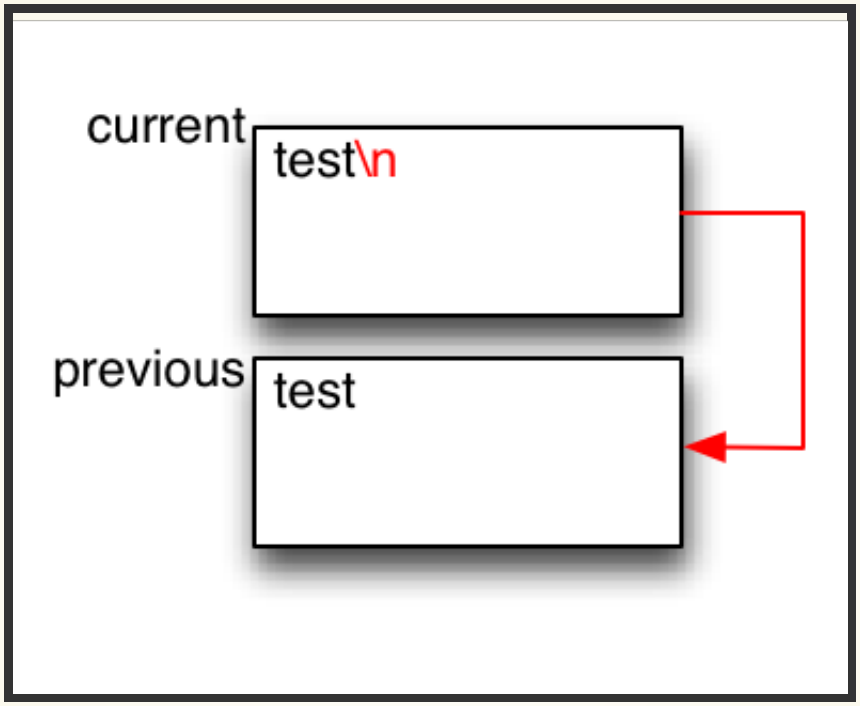
PARSING HBOOT COMMANDS

```
char current_cmd[256];
char previous_cmd[256];
void hboot_command_line() {
  unsigned int len = 0;
  char* buf = current_cmd;
  char* current_char;
  while (1) {
    if (!usb_read(buf, 1)) //read one character
      break;
    current_char = *buf;
    switch (current_char)
    {
      case '\n':
        *buf = '\0';            //breakpoint 1
        strcpy(previous_cmd, current_cmd);   //breakpoint 3
        hboot_handle(current_cmd);
      case '\t':
        *buf = ' ';             //breakpoint 2
        strcpy(buf, previous_cmd);
        len = strlen(buf)
        buf += len;
      //...
```

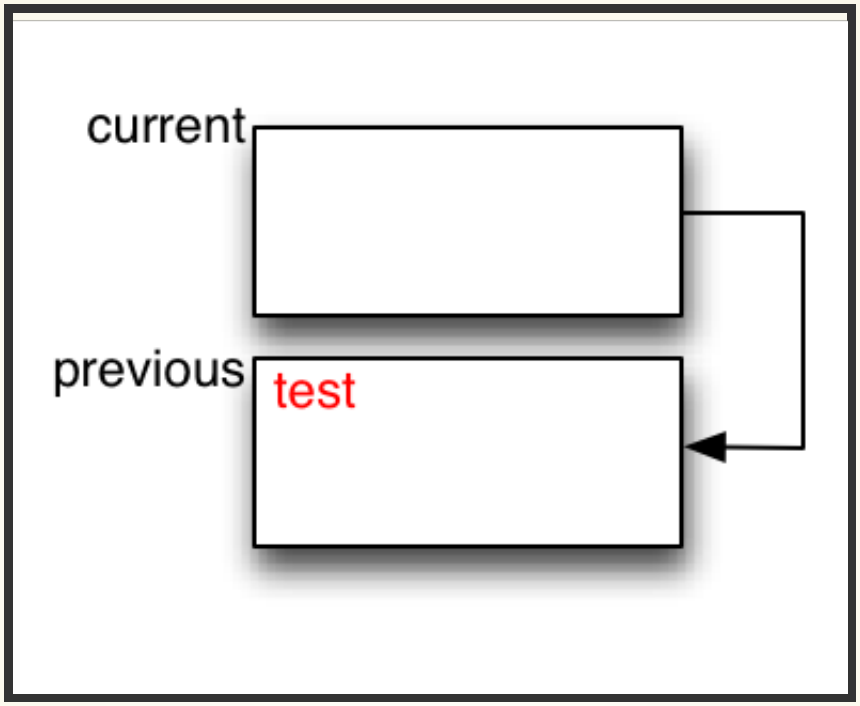
PARSING HBOOT COMMANDS



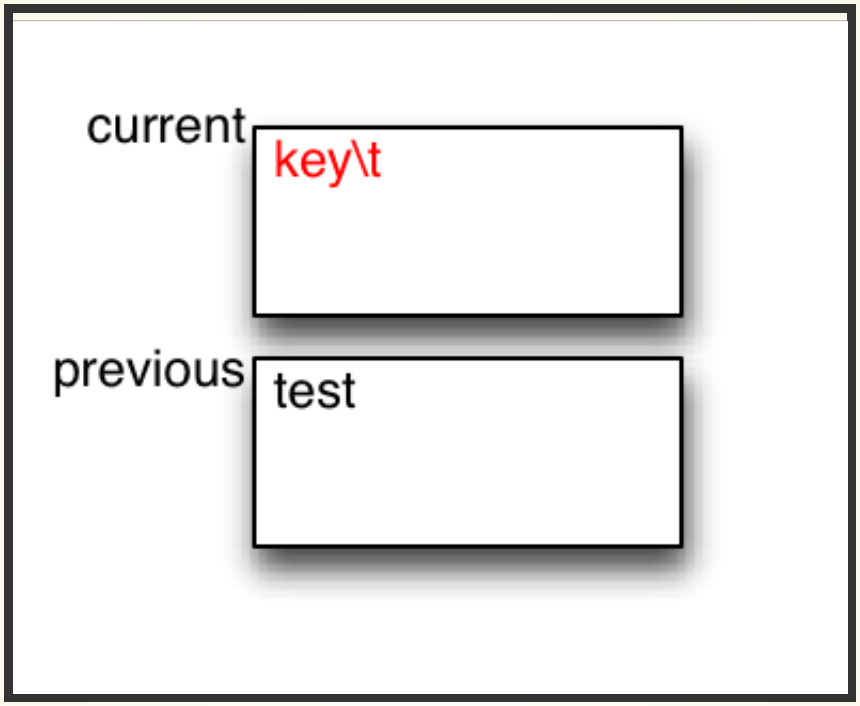
PARSING HBOOT COMMANDS



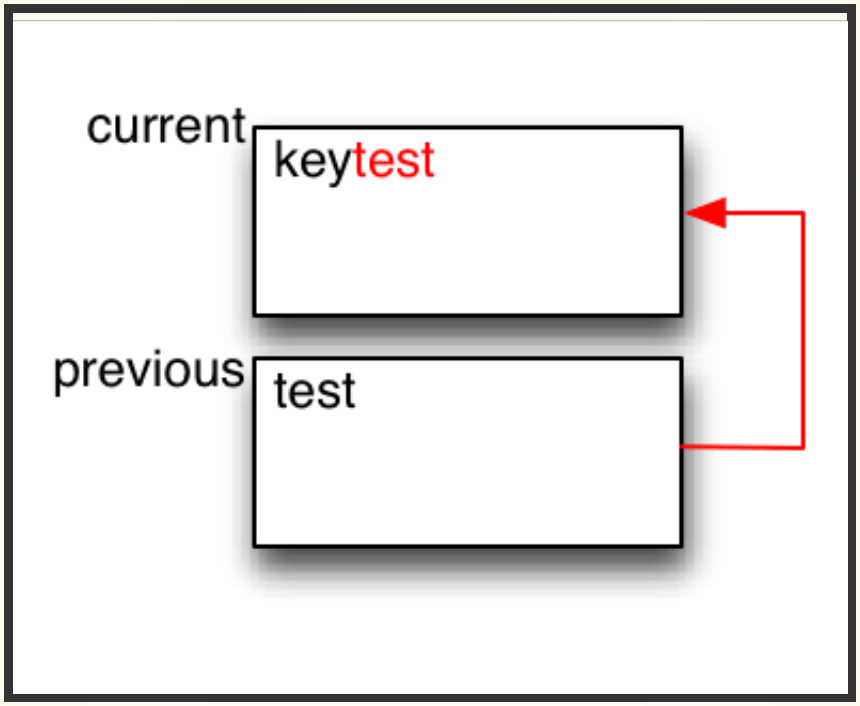
PARSING HBOOT COMMANDS



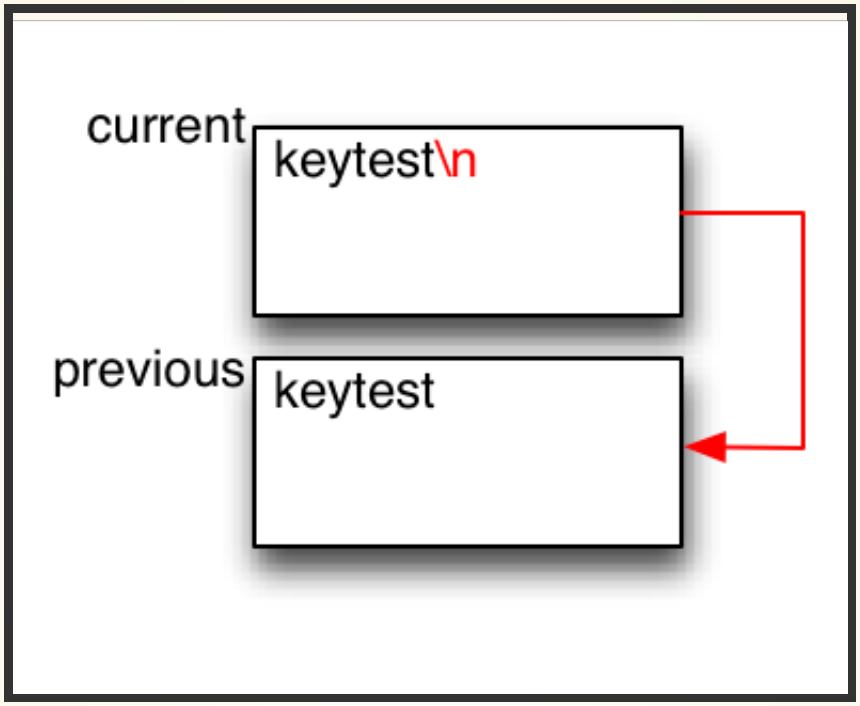
PARSING HBOOT COMMANDS



PARSING HBOOT COMMANDS



PARSING HBOOT COMMANDS



PARSING HBOOT COMMANDS

- Read one character at a time into a 256-byte buffer
- If "end of command" (\n)
  - Save first buffer into second buffer and handle command
- If "tabulation" (\t)
  - Copy second buffer at first buffer end

- Idea behind '\t' feature
  - First buffer: current command
  - Second buffer: saved command
  - Append previous command to prompt with tabulation

PROBLEM IN COMMANDS PARSING

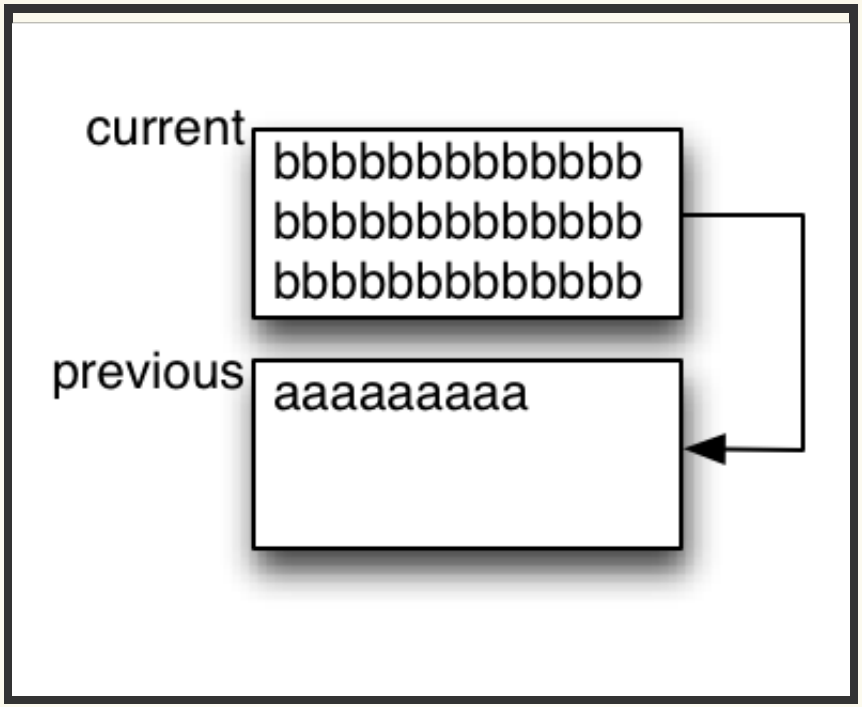
- When using tabulation
  - No check that current command buffer big enough to append previous command
- Overflow the buffer of the current command

- What is really happening? ⇒ Using our debugger
  - Note: Debugger conflicts with command console, need to switch between them

DEMO

Analyzing the problem with our debugger

PARSING HBOOT COMMANDS



- Destination buffer increased when *strcpy*
- Source and destination buffer adjacents
  - Source buffer increases as well ⇒ *strcpy* loops infinitely :(

AGENDA

1. Basics
2. Revolutionary vulnerability
3. HBOOT debugger
4. Simple bug
5. Conclusion

CONCLUSION

- Functional debugger
- Reverse engineering to find a bug
  - Using the debugger ⇒ not exploitable on its own
- HBOOT command parsing improvable

- Debugger source code should be released soon

FUTURE WORK

- Revolutionary vulnerability fixed on recent devices (eg: HTC One with HBOOT 1.44)
- Port debugger using another vulnerability (eg: rumrunner)
  - Look at how rumrunner works
  - Buy a HTC One :)
- Continue our analysis of HBOOT

THANK YOU FOR YOUR ATTENTION

cedric.halbronn@sogeti.com - @saidelike
nicolas.hureau@sogeti.com - @kalenz