PatchDroid: Scalable Third-Party Security Patches for Android Devices

29th ACSAC 2013

Collin Mulliner, Jon Oberheide, William Robertson, Engin Kirda
{crm,wkr,ek}[at]ccs.neu.edu, jono[at]duosecurity.com

Android Security

- Android platform contains security vulnerabilities
  - New vulnerabilities are discovered all the time
- Android has built-in update mechanism
  - Over-the-Air (OTA) updates
  - No desktop computer needed
- Google patches a bug
  - Update arrive at Nexus devices (Google devices)
  - Patches are pushed to AOSP
  - Manufacturers are notified

Android Security

- Android platform contains security vulnerabilities
  - New vulnerabilities are discovered all the time

- Android has built-in update mechanism
  - Over-the-Air (OTA) updates
  - No desktop computer needed

- Google patches a bug
  - Update arrive at Nexus devices (Google devices)
  - Patches are pushed to AOSP
  - Manufacturers are notified

- **Unfortunately, only few devices receive updates!**

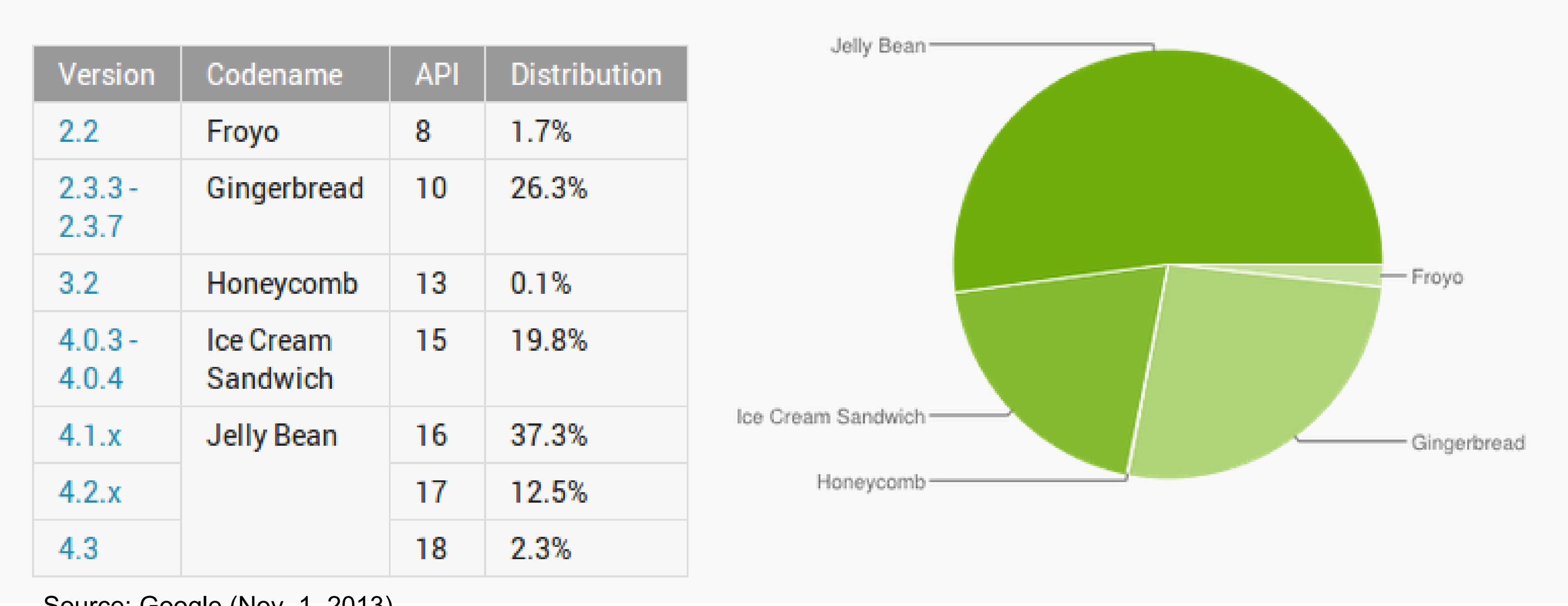
Missing Updates

- Manufacturer
  - Stop supporting devices after about 18 months
  - Manufacturer specific bugs

- Carrier
  - Customize firmware
  - Delay updates due to customization efforts
  - Do not update at all

- Result → many devices run out of date software
  - Software that contains publicly known vulnerabilities

Android Platform Version Diversity



Version	Codename	API	Distribution
2.2	Froyo	8	1.7%
2.3.3 - 2.3.7	Gingerbread	10	26.3%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	19.8%
4.1.x	Jelly Bean	16	37.3%
4.2.x		17	12.5%
4.3		18	2.3%

Source: Google (Nov. 1, 2013)

Patching Vulnerabilities on Android Devices

- Overlooked problem until now
  - Google and manufacturers' duty

- Only solution so far is 3rd party firmware
  - Available for limited number of devices only
  - Manual process, no automated follow-up update

- Platform diversity is the key problem
  - Large number of different devices + software versions
  - Any solution has to address these problems

Challenges

- No access to source code
  - AOSP ≠ code running on devices
  - Modifications by the manufacturer
- Issue with modification of system files and partitions
  - Modified binaries might prevent system from booting
  - Cannot add/replace files on signed partitions
- Scalability vs. Testing
  - Too many different devices and OS versions
  - Patches need to be decoupled from the source code

Contributions

- **PatchDroid**: third-party security patches for Android
  - Includes attack detection and warning mechanism
- Scalable
  - Independent from device and Android version
  - Support for managed Dalvik bytecode and native code
- Reliable
  - No permanent modification (no bricked devices)
- Usable in practice
  - No noticeable overhead (no device slow down)
  - Does not rely on access to source code

Overview

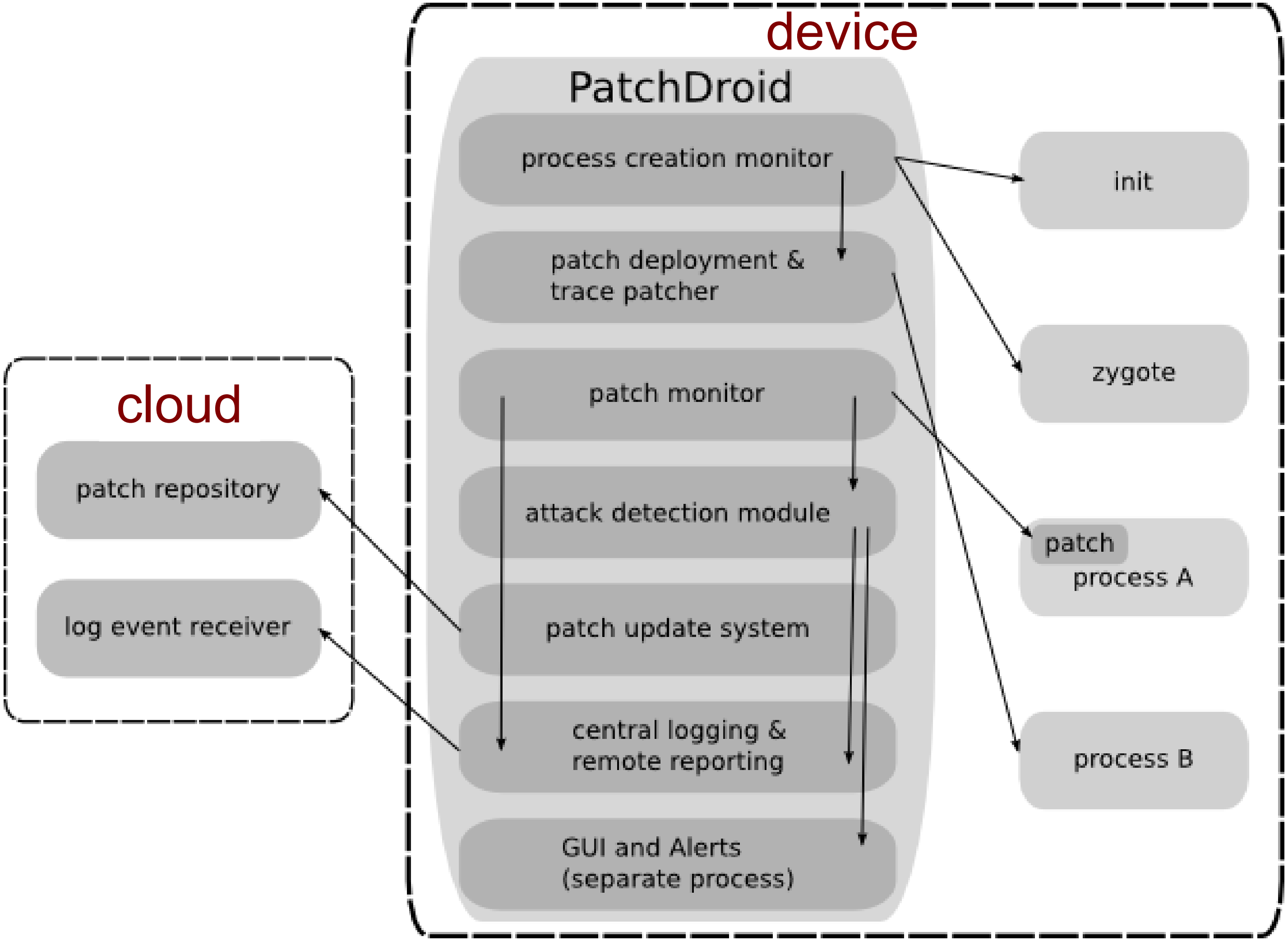
- Design
- Patches and Patching
- Implementation
- System Evaluation
- Case Study: MasterKey

The PatchDroid System

- In-memory patching at runtime
  - Need to patch processes at startup
    - Before process executes vulnerable code
    - Monitor system for new processes
  - No need to modify system files or system partitions

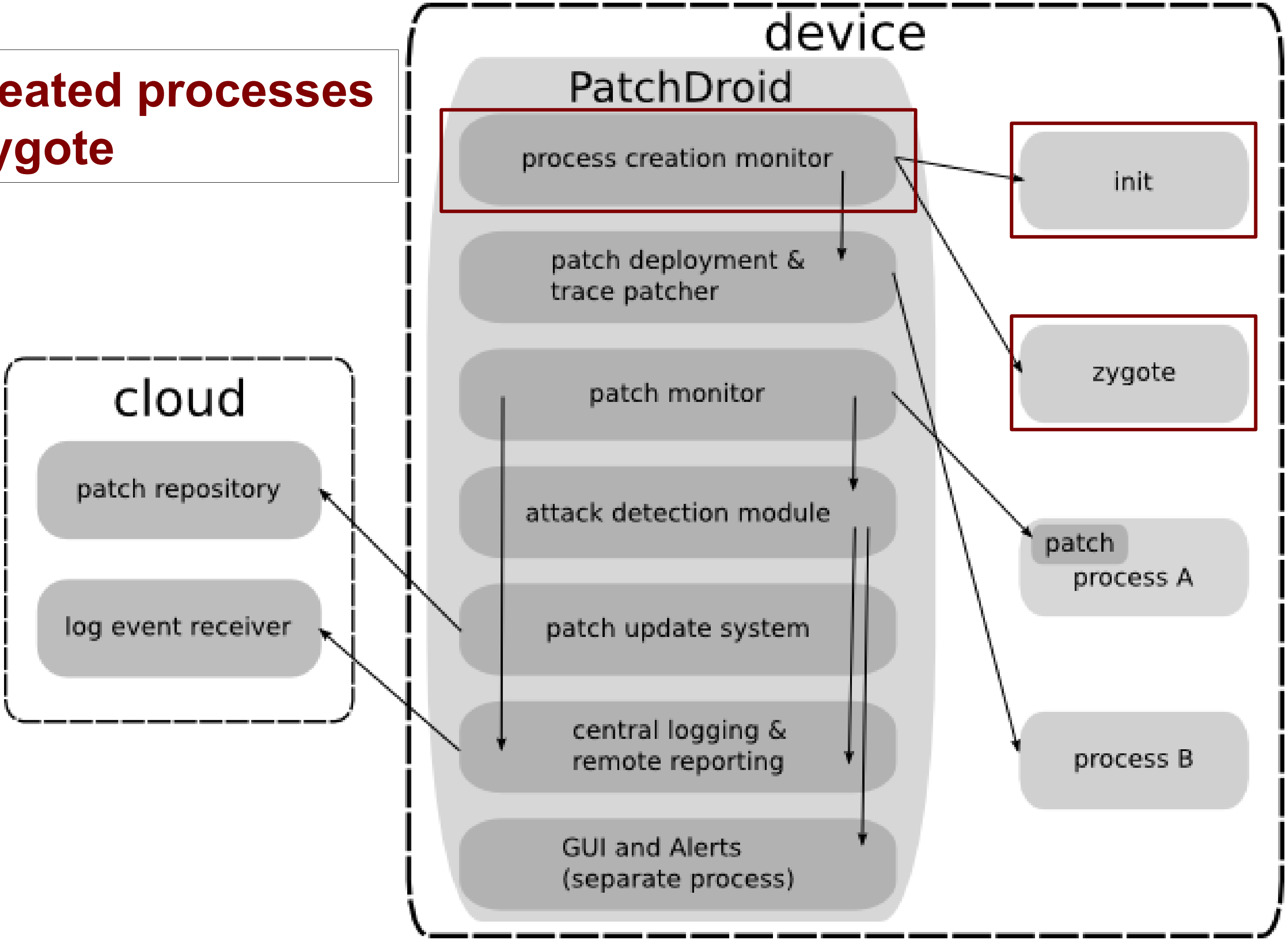
- Patches as independent code
  - Self-contained shared library
  - Patching via function hooking
  - No access to original source code required
  - Scale across different OS versions

PatchDroid : Architecture



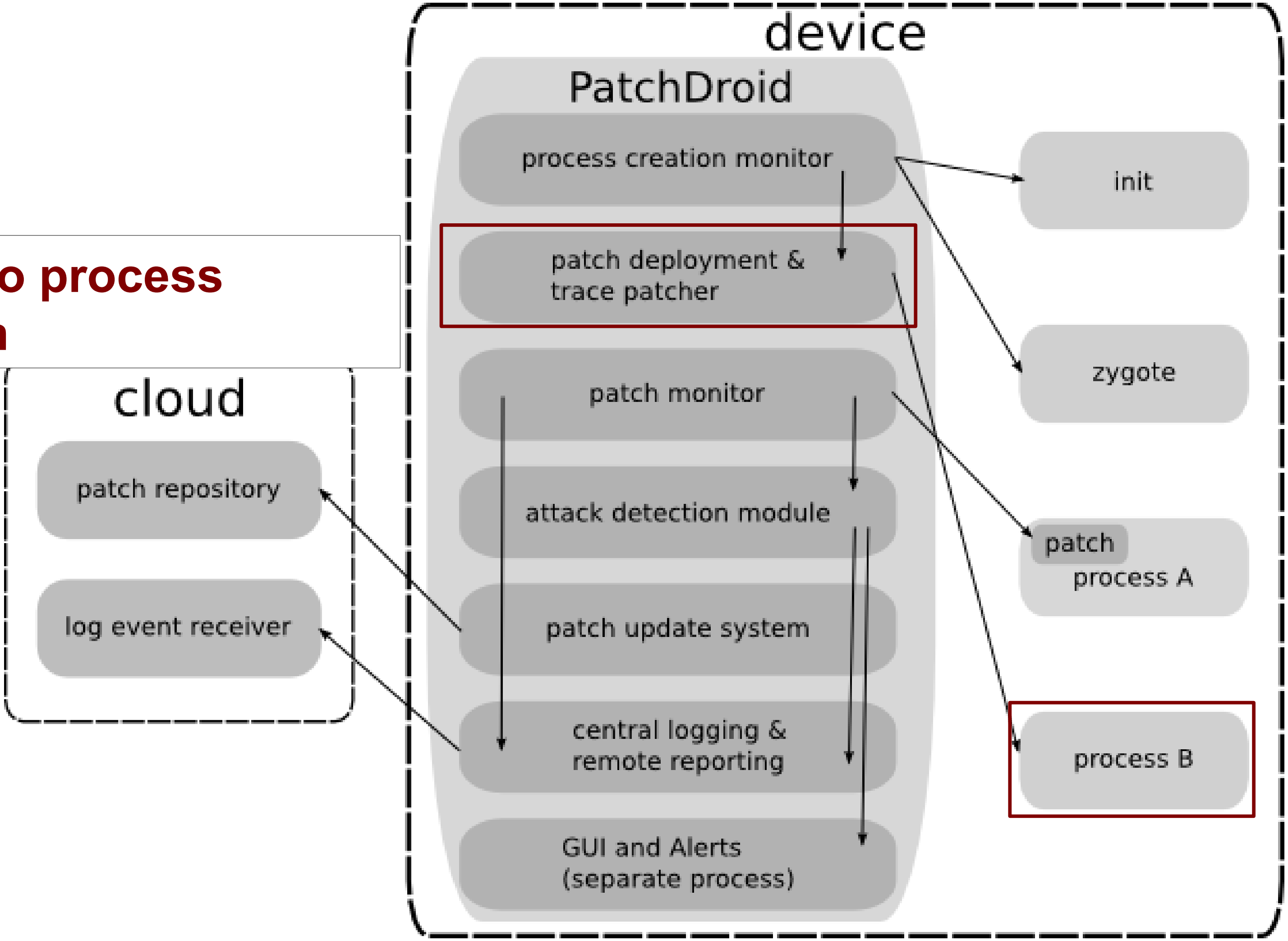
PatchDroid : Architecture

Identify newly created processes
- trace init and zygote



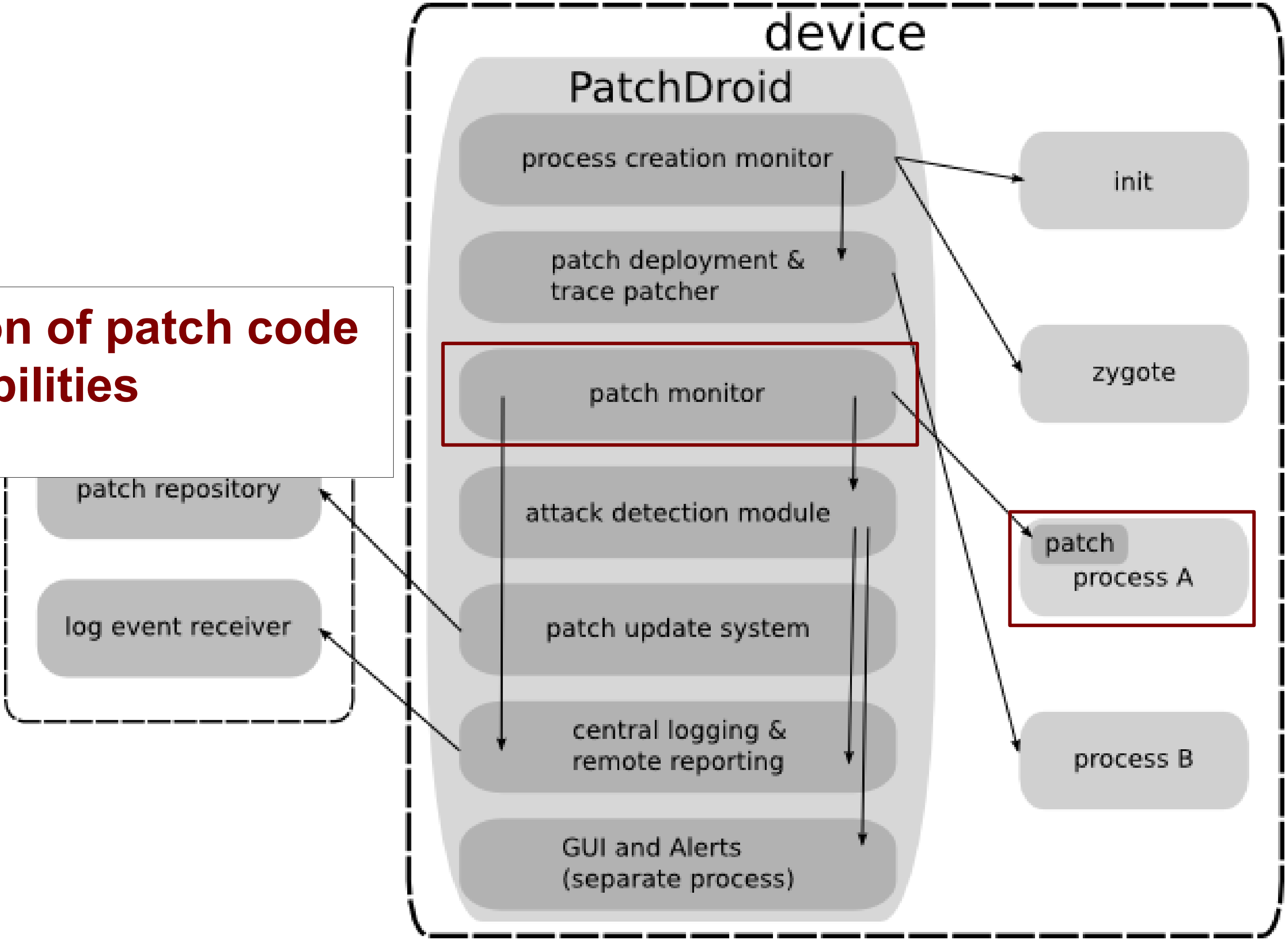
PatchDroid : Architecture

Deploy patch into process
- library injection



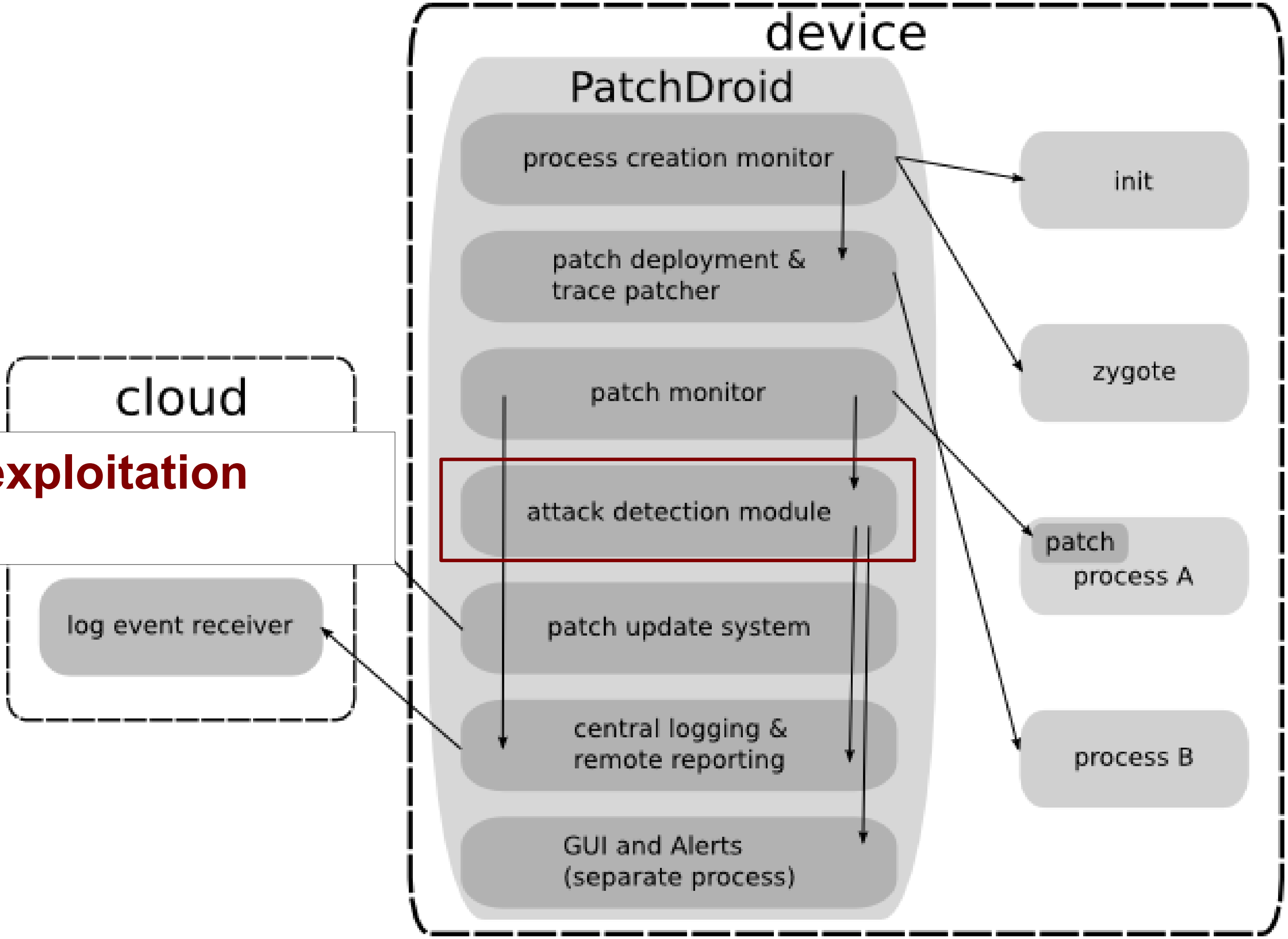
NEU SECLAB

PatchDroid : Architecture



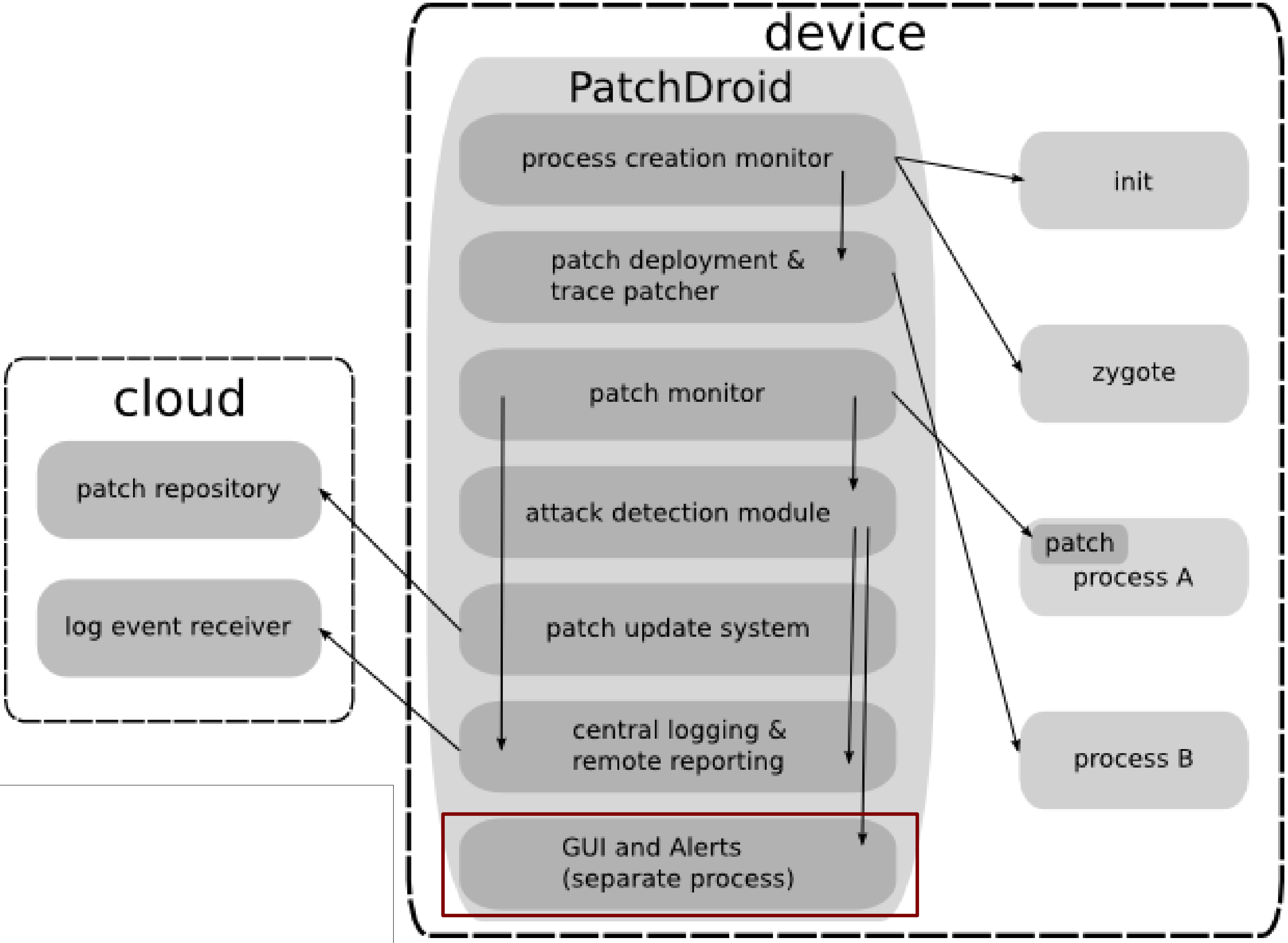
Monitor execution of patch code
- check for instabilities
- collect logs

PatchDroid : Architecture



Analyze log for exploitation attempt

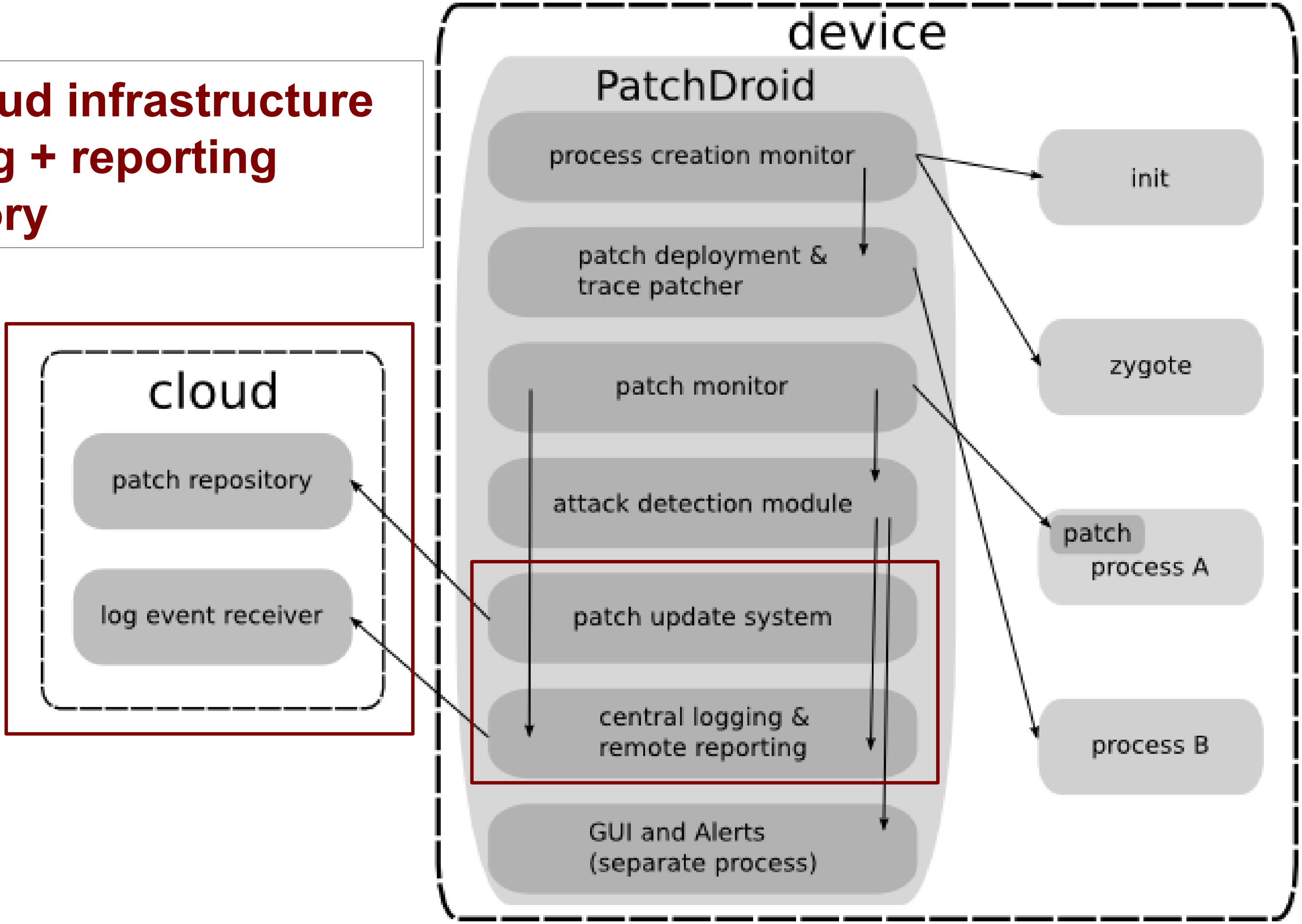
PatchDroid : Architecture



PatchDroid App
-GUI
-display alerts

PatchDroid : Architecture

PatchDroid cloud infrastructure
-central logging + reporting
-patch repository

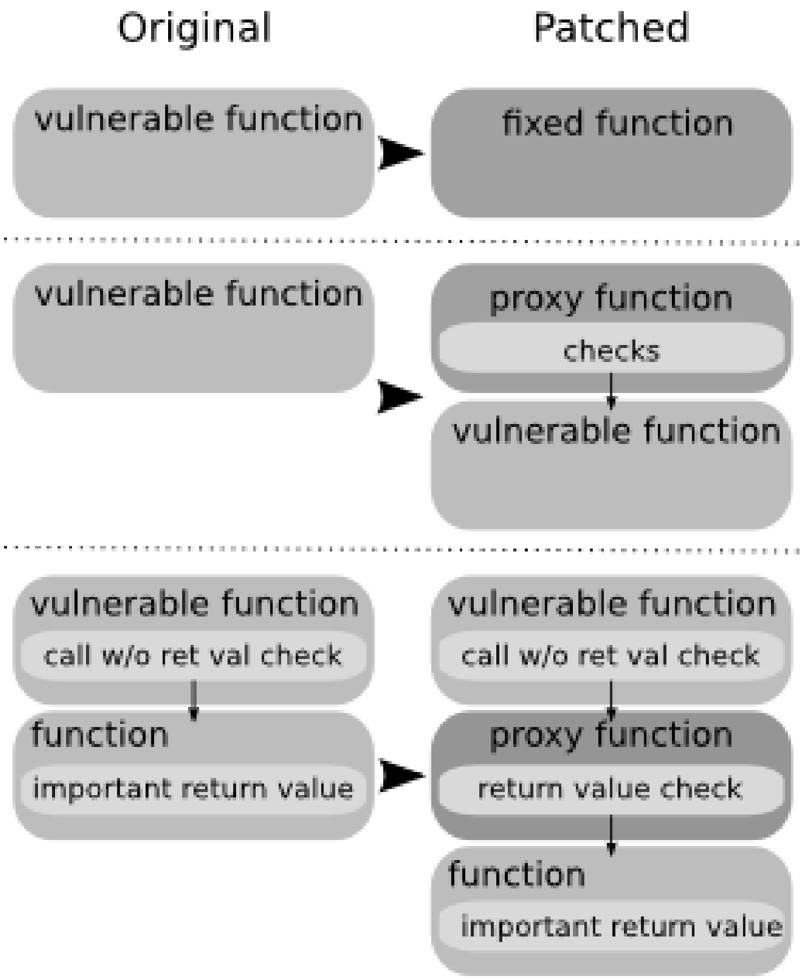


Anatomy of a Patch

- Replacement for vulnerable function
  - Equivalent code that does not contain the vulnerability
  - Wrapper that adds input/output sanitization
- Installation
  - Hook vulnerable function(s)
    (original function needs to be kept in working condition)
- Communication link
  - Read configuration parameters
  - Write log messages

Patching Strategies

- Function replacement
- Proxy function
- Failed return value check



Example: Failed return value check

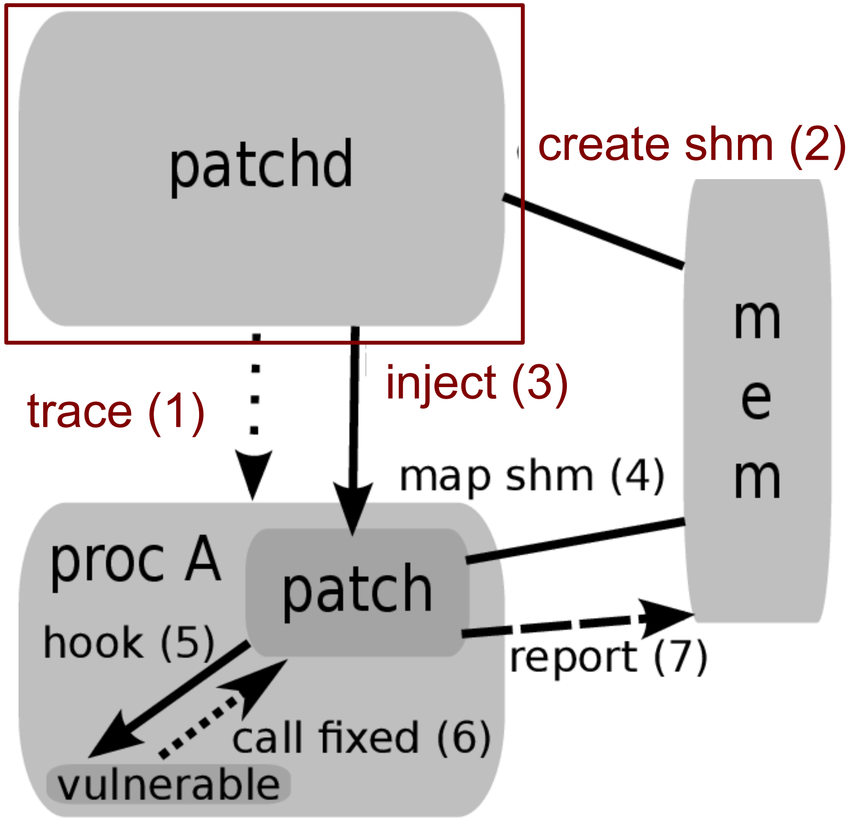
- int res = setuid(nobody);
  - res == 0 → success, privileges dropped
  - res == -1 → failure, privileges NOT dropped

- Missing check of result in zygote
  - fork() until setuid() failed due to resource limit
    → new process stays root!

- Patch: wrap setuid()
  - check result
  - terminate if res != 0

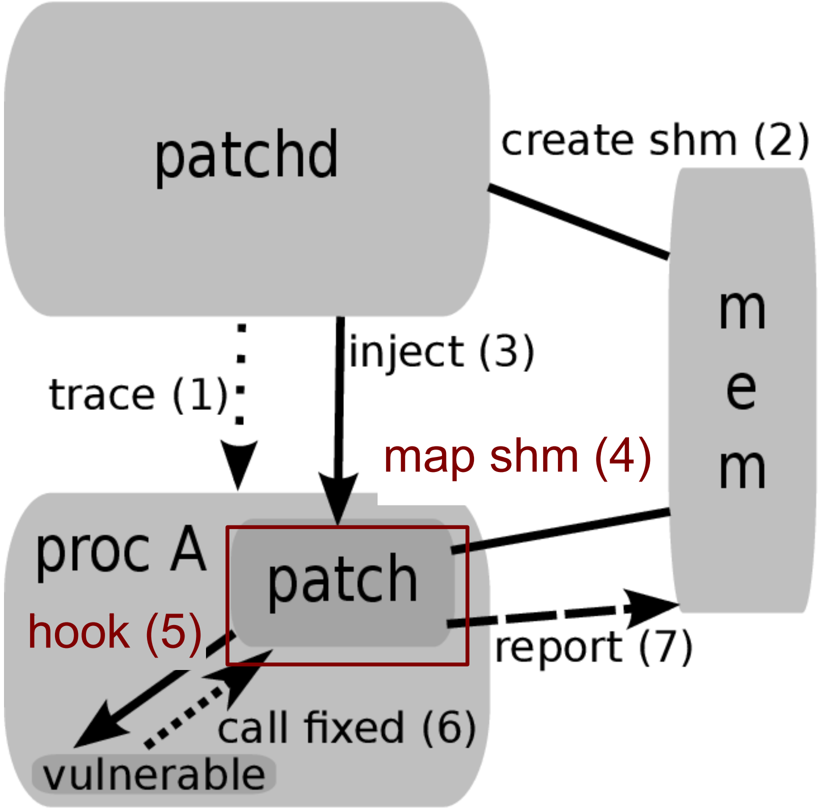
Patch Life Cycle

- Deployment
  - trace target process
  - setup communication
  - inject patch library



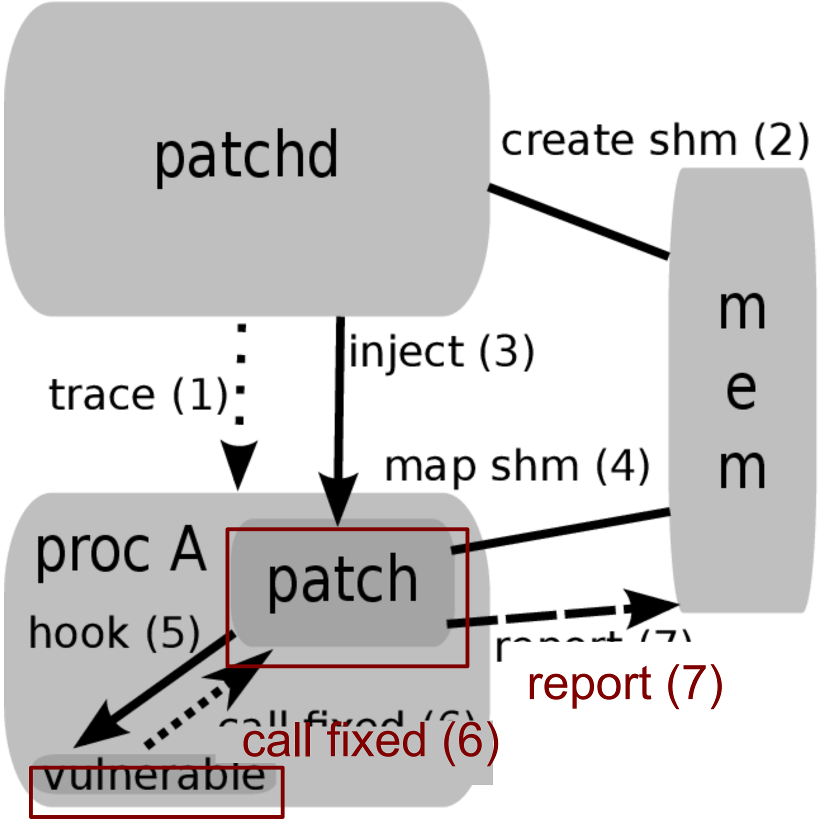
Patch Life Cycle

- Installation
  - connect communication
  - hook function

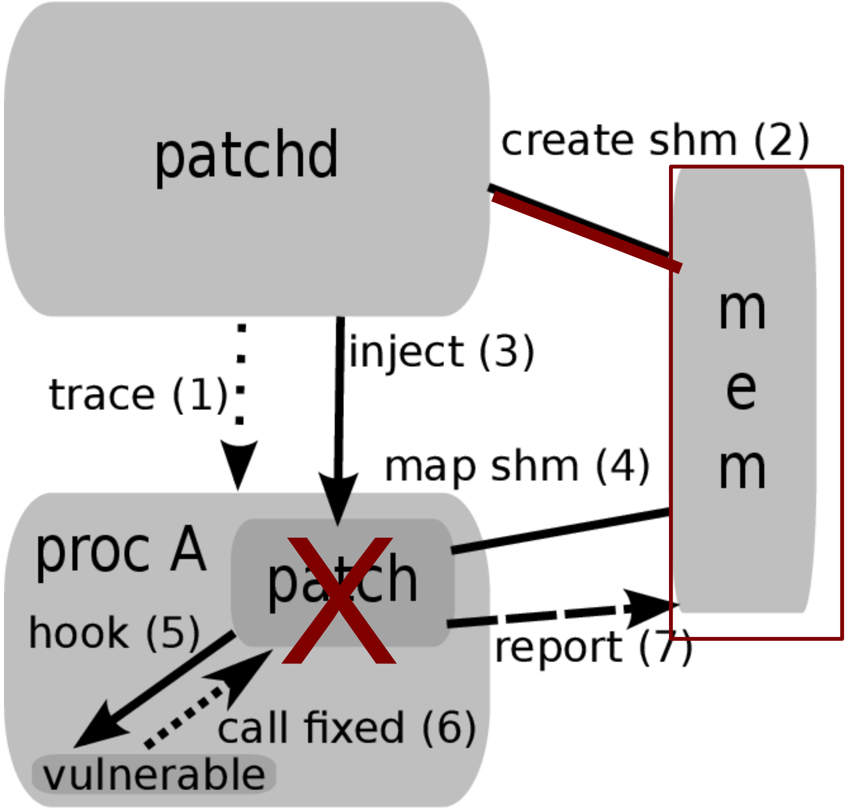


Patch Life Cycle

- Fixed function is called
  - log and report attack
  - collect patch telemetry
  - (call original function)



Patch Life Cycle



- Patch failure
  - detected using telemetry
  - failing patch is removed from system

- <u>Enables scalable testing of patches in the field</u>

Implementation

- *patchd:* the patch daemon
  - Monitor system for newly created processes
  - Inject patches into processes
  - Monitor patched processes

- PatchDroid Application
  - User interface
  - Helper service
  - Attack notification

- Patches
  - 3 patches for privilege escalation (native code)
  - 1 patch for permission leak bug (Dalvik code)

Patch Creation

- Extract patch from AOSP and **transform** to PatchDroid
  - Apply patch strategy best suited for vulnerability

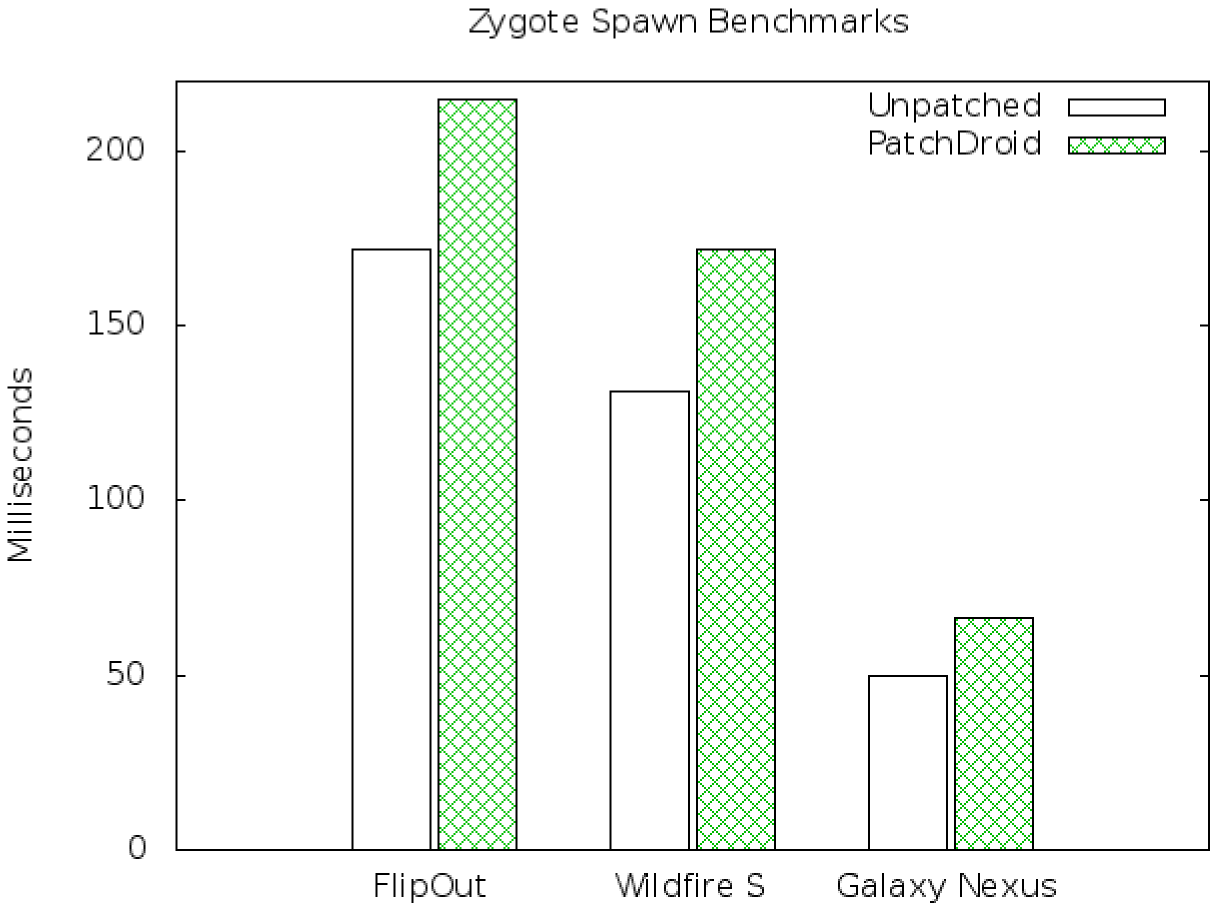
- Develop custom patch
  - Bug known but no patch available

Evaluation

- System performance
  - Low overhead during process creation
  - No runtime overhead
- Functional testing
  - Patch vs Exploit
- User trials
  - Users run PatchDroid
  - Try exploiting known vulnerabilities (details in the paper)

Overhead – creating new process

- One time hit at process creation



Patch vs Exploit

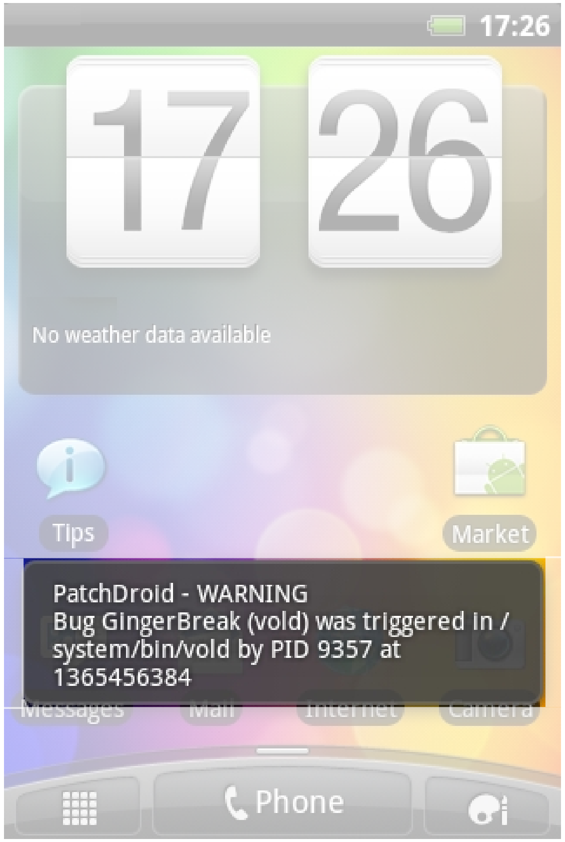
- Privilege escalation vulnerabilities (root exploits)
  - Zimperlich
  - GingerBreak
  - ZergRush

- Permission leak
  - local SMS spoofing (Dalvik)

- All patches prevent exploitation on the affected devices
  - PatchDroid warns the user about attack

Attack Detection & Warning

- GingerBreak on Android 2.3



PatchDroid - WARNING
Bug GingerBreak (vold) was triggered in /system/bin/vold by PID 9357 at 1365456384

Case Study: MasterKey Bug(s)

- Bug(s) in handling of APK files
  - APK can be modified w/o breaking the signature

- MasterKey can be used for privilege escalation
  - Modify APK signed with platform/manufacturer key
  - Works on all devices from manufacturer

- Bug in manifested in Dalvik bytecode
  - First privilege escalation vulnerability in Dalvik code

- Present in all Android version until 4.3
  - Affected all Android devices at the time

Patching MasterKey

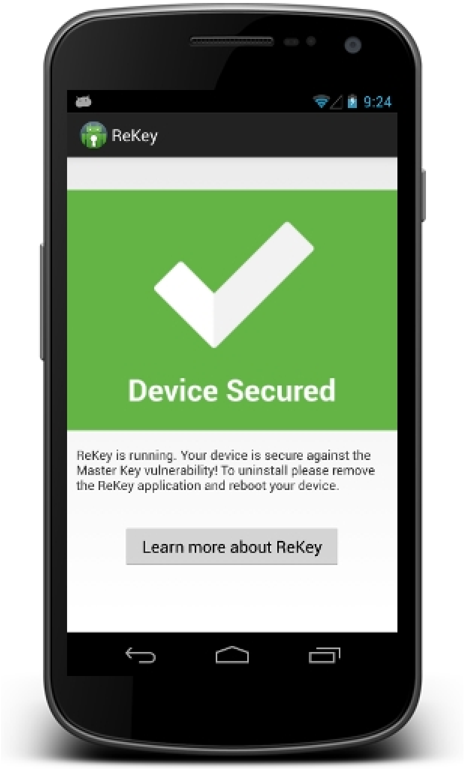
- Patch Strategies
  - Missing return value check
  - Proxy function

- Fast implementation and testing
  - Initial version took only three (3) hours

- We wanted to release the patch to the general public
  - Provide possibility to protect user's devices
  - At this time the paper was still under review

ReKey

- Limited version of the PatchDroid system
  - Only the MasterKey bug(s)
- Released ReKey on the Google Play store
  - July 16th
- Currently 12,000 users
  - Only works on rooted devices
- ReKey your device!
  - **http://www.rekey.io**



ReKey is running. Your device is secure against the Master Key vulnerability! To uninstall please remove the ReKey application and reboot your device.

Conclusions

- We are the first to address security patching on Android

- With PatchDroid we show that
  - third-party patching is possible without source code
  - patch development scales across different devices

- **PatchDroid**
  - supports Dalvik and native code
  - no noticeable performance overhead
  - no impact device stability
  - safe against accidentally “bricking” devices

- Public release of ReKey was a huge success

EOF

Thank you!

Questions?

http://www.patchdroid.com