Mobile Security

Dr. Eleonora Losiouk
Department of Mathematics
University of Padua
elosiouk@math.unipd.it
https://www.math.unipd.it/~elosiouk/

Università degli Studi di Padova

SPRITZ Security & Privacy Research Group

Dipartimento Matematica

Classes of vulnerabilities

Classes of vulnerabilities

- There are many classes of vulnerabilities
- Here we discuss the ones related to mobile devices
  - The list is not exhaustive...
- We will discuss them by "**attack surface**"

Attack Surface Enumeration

- From high-level to low-level
  - the user
  - apps (third-party apps and system apps)
  - system components / operating system
  - hardware (RAM)

Attack Surface Enumeration

- From high-level to low-level
  - the user
  - apps (third-party apps and system apps)
  - system components / operating system
  - hardware (RAM)

Attacking the user

- Main attack vector against user: social engineering

Attack Surface Enumeration

Università degli Studi di Padova

- From high-level to low-level
  - the user
  - apps (third-party apps and system apps)
  - system components / operating system
  - hardware (RAM)

Attacking apps

- Many things can go wrong in many different ways

- Two main aspects
  - What
    - Attacker may *abuse sensitive resource/permission* the app has access to
    - Attacker may *leak sensitive information* the victim app has access to
  - How
    - How can an attacker interact with a target app?
    - *Entry point enumeration*

Attacking apps

- App connects to network backend

Attacking apps

Università degli Studi di Padova

- App connects to network backend
- Dynamic code loading

Attacking apps

- App connects to network backend
- Dynamic code loading
- Cryptographic vulnerabilities

Confused Deputy Problem

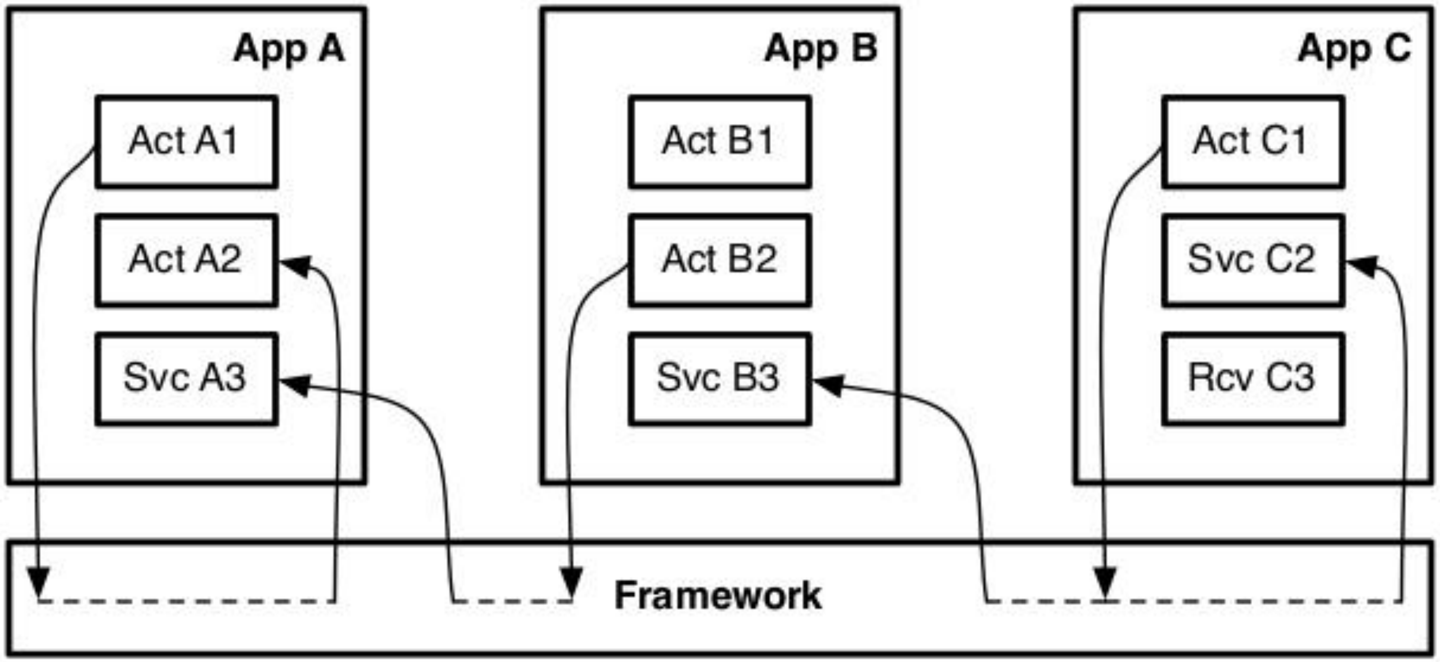
Università degli Studi di Padova

- Consider app A
  - it has access to sensitive information
  - it contains functionality using sensitive permissions

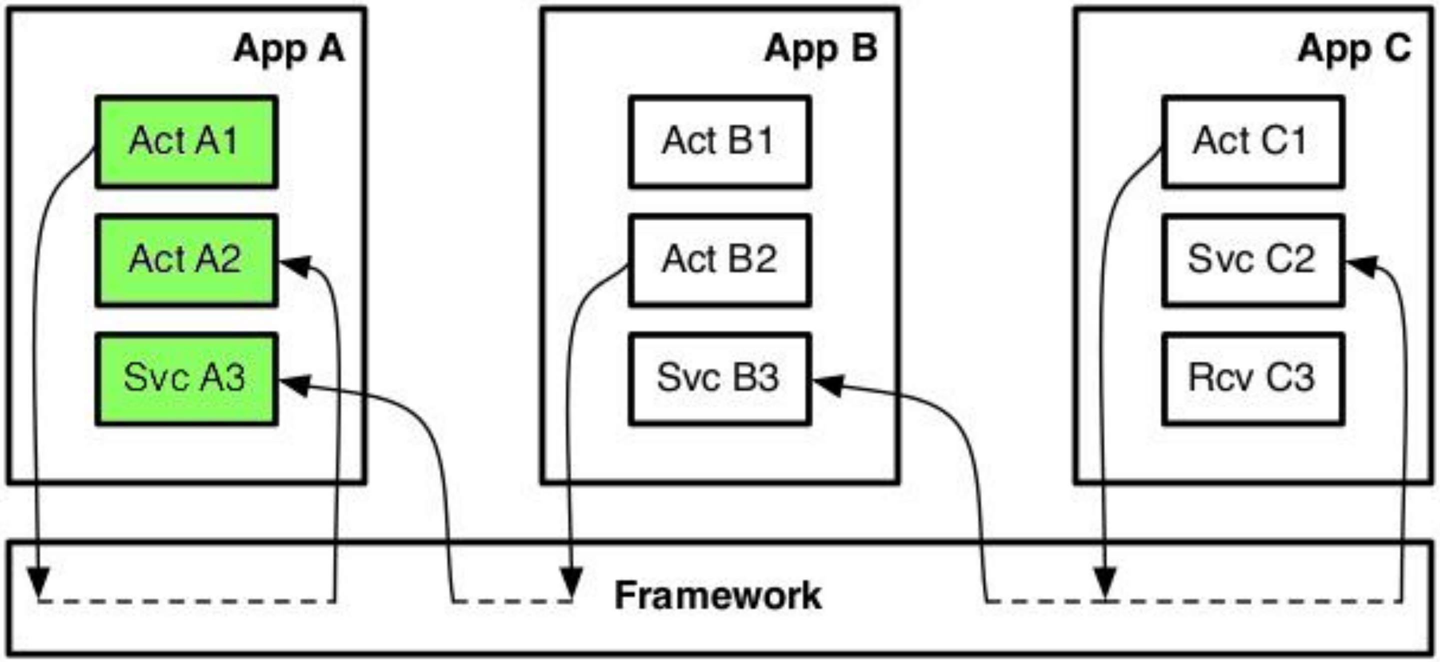
- Confused Deputy Problem arises when App A doesn't properly protect such sensitive aspects from other apps

Confused Deputy Problem

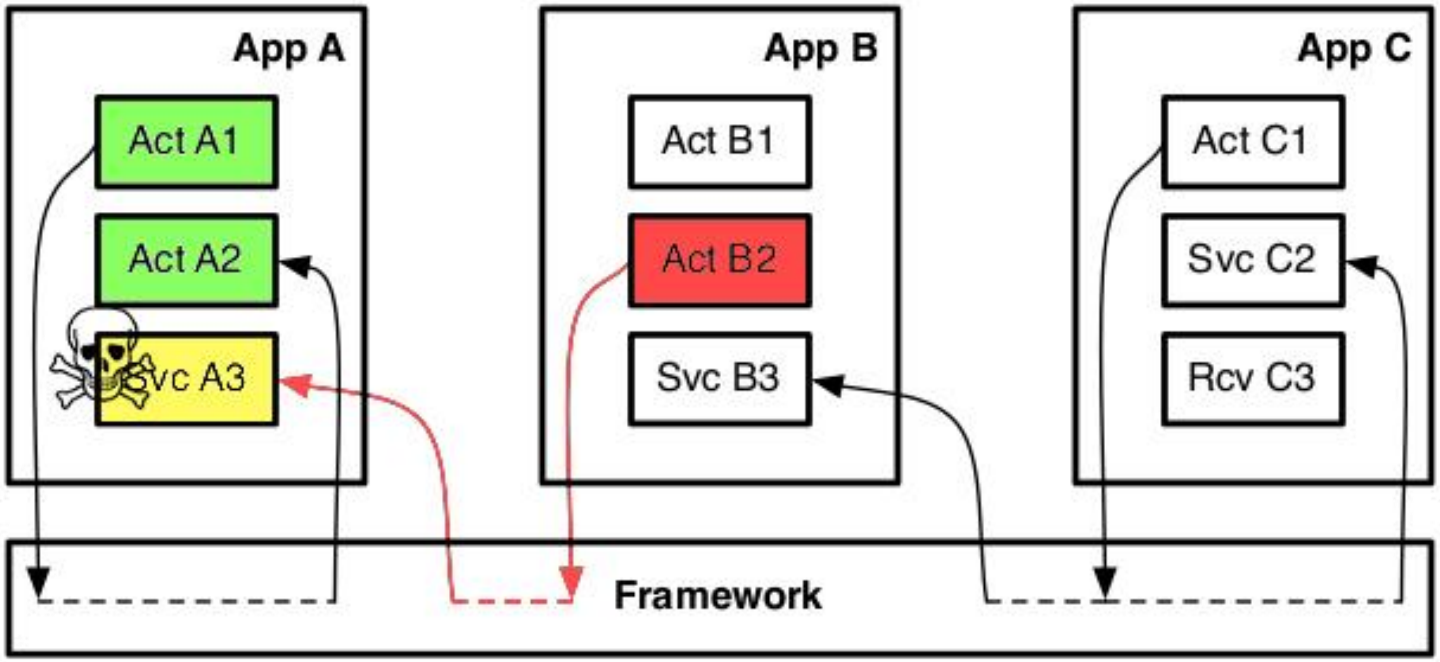
Università degli Studi di Padova



Confused Deputy Problem



Confused Deputy Problem



Component Hijacking

- Component Hijacking (CH) attacks aim at gaining unauthorized access to protected resources of an app through its exported components

Permission/Capability Leak

- Permission Leak
  - Instance of confused deputy problem
- Example: app B can "use" permission X via app A

Content Leaks & Pollution

- Content providers are wrappers around databases
- What if these databases allow "too much" access to external apps?
  - Leak: disclose various types of private in-app data
  - Pollute: manipulate security-sensitive in-app settings or configurations

Overpermissioning

- Overpermissioning
  - App requires permission X even if it does not need it
- Not a bug per-se, but it represents an unnecessary risk
  - If affected by confused deputy problem, permission X could be abused

Zip Path Traversal

- Many libraries/frameworks are affected by a "unsafe unzip path traversal" problem
- A zip file can contain a relative ../../evil.sh file path
  - When unzipped, it can overwrite files in different directories
  - ⇒ File write to code execution via cached DEX overwrite
- Concrete examples
  - Remote Code Execution on Samsung Keyboards
  - Zip Slip Vulnerability

Zip Path Traversal

Università degli Studi di Padova

- Remote Code Execution on Samsung Keyboard

```
GET http://skslm.swiftkey.net/samsung/downloads/v1.3-USA/az_AZ.zip
     ← 200 application/zip 995.63kB 601ms
```

```
root@kltevzw:/data/data/com.sec.android.inputmethod/app_SwiftKey/az_AZ # ls -l
        -rw------- system   system     606366 2015-06-11 15:16 az_AZ_bg_c.lm1
        -rw------- system   system    1524814 2015-06-11 15:16 az_AZ_bg_c.lm3
        -rw------- system   system        413 2015-06-11 15:16 charactermap.json
        -rw------- system   system         36 2015-06-11 15:16 extraData.json
        -rw------- system   system         55 2015-06-11 15:16 punctuation.json
```

```
$ unzip -l evil.zip
         Archive:  evil.zip
          Length       Date    Time     Name
        ---------  ---------- -----    ----
                5  2014-08-22 18:52    ../../../../../../../../data/payload
        ---------                      -------
                5                      1 file
```

Zip Path Traversal

- Remote Code Execution on Samsung Keyboard

```
root@kltevzw:/data/dalvik-cache # /data/local/tmp/busybox find . -type f -group 1000
        ./system@framework@colorextractionlib.jar@classes.dex
        ./system@framework@com.samsung.device.jar@classes.dex
        ./system@framework@com.quicinc.cne.jar@classes.dex
        ./system@framework@qmapbridge.jar@classes.dex
        ./system@framework@rcsimssettings.jar@classes.dex
        ./system@framework@rcsservice.jar@classes.dex
        ./system@priv-app@DeviceTest.apk@classes.dex
```

Critical: the keyboard app could NOT be uninstalled

This is code!

Native Code

- Apps can have native code components, written in C/C++
- C/C++ code can be vulnerable to a number of memory corruption vulnerabilities
    - Buffer overflows, dangling pointers, use after free, type confusion, etc.
- If an attacker's input can "reach" these components...
    - ⇒ then these vulns come into play

More apps' vulns

- We have seen many more potential vuln patterns during the course!
- Given API X, it's likely there is a way to misuse it...

Attack Surface Enumeration

- From high-level to low-level
  - the user
  - apps (third-party apps and system apps)
  - system components / operating system
  - hardware (RAM)

Attacking the system

- Bugs can affect the Android framework / OS itself
- There are many: tens of vulns every month!
- They can affect different components
  - Framework, media framework, system, kernel components, Qualcomm components

Unsafe Self-Update

- Android updating mechanism affected by PileUp flaws
- An application requires a permission that does not exist yet
- Privilege escalation through OS updating

L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang. Upgrading Your Android, Elevating My Malware: Privilege Escalation Through Mobile OS Updating. In IEEE Symposium on Security and Privacy, 2014.

Attacking the system

- Vulns in some components usually lead to higher severity

- Example: the Media framework
    - The Media framework processes, among many things, images
    - Bugs are often in media parsing
    - Media parsing is often "triggerable" remotely
        - MMS, email, visiting a website
    - The mere fact that these code components can be reachable by a remote attacker is already enough to make these bugs more dangerous

Stagefright bug

- Stagefright (August 2015)
  - Main Android’s media processing library
- Several critical vulnerabilities in media parsing
  - End result: remote code execution by sending an MMS
- Biggest security vulnerability in Android at that point
  - This is the bug that pushed Google to create monthly security bulletins

Stagefright bug


```
status_t MPEG4Source::parseChunk(off64_t *offset) {
    uint64_t chunk_size = ntohl(hdr[0]); // attacker-controlled!
    size_t size = 0;

    if (!mLastTrack->meta->findData(
            kKeyTextFormatData, &type, &data, &size)) {
        size = 0;
    }

    uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
    [...]

    if (size > 0) {
        memcpy(buffer, data, size);
    }
}
```

size_t

uint64_t

The parameter of the "new" operator has type size_t

Stagefright bug

- On 32-bit architectures, “size + chunk_size” may be truncated (because size_t is 32-bit)

```
    if (!mLastTrack->meta->findData(
              kKeyTextFormatData, &type, &data, &size)) {
         size = 0;
    }

    uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
    [...]

    if (size > 0) {
         memcpy(buffer, data, size);
    }
}
```

size_t

uint64_t

The parameter of the “new” operator has type size_t

Stagefright bug

- On 32-bit architectures, “size + chunk_size” may be truncated (because size_t is 32-bit)
- On 64-bit architectures, “size + chunk_size” may overflow

```
    if (!mLastTrack->meta->findData(
            kKeyTextFormatData, &type, &data, &size)) {
        size = 0;
    }

    uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
    [...]

    if (size > 0) {
        memcpy(buffer, data, size);
    }
}
```

`sizeof(buffer) < sizeof(data)`

size_t

uint64_t

Buffer overflow!

The parameter of the “new” operator has type size_t

Baseband vulnerabilities

- A walk with Shannon: A walkthrough of a pwn2own baseband exploit - Amat Cama

- Memory corruption that
  - Can be triggered by a phone connecting to a malicious base station
  - Leads to remote code execution in the baseband processor
    - Remote ~ Proximal attacker

Bootloader vulnerabilities

- Bootloader is a program, and it can contain bugs as well!

- Bootloader bugs can lead
- Execute arbitrary code (as part of the bootloader)
    - Bypass of secure boot ⇒ bypass of chain of trust
    - Permanent denial-of-service

Attack Surface Enumeration

- From high-level to low-level
  - the user
  - apps (third-party apps and system apps)
  - system components / operating system
  - hardware (RAM)

Rowhammer bug

- Problem affecting DRAM cells
  - Memory cells leak their charges (when properly "stimulated")
  - Net effect: a bit flips in memory

Rowhammer bug

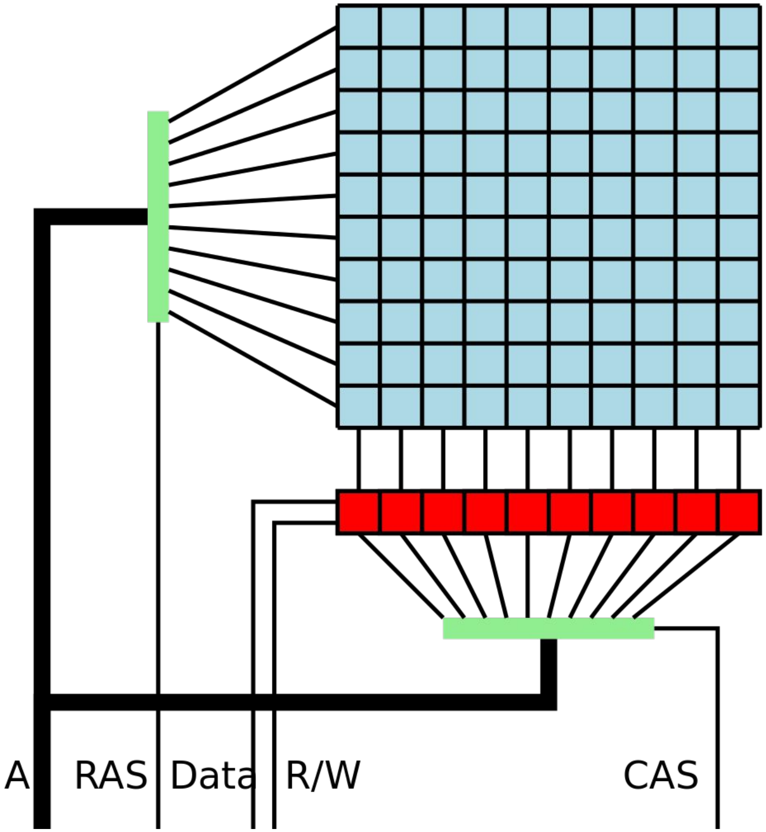


Image from wiki

Rowhammer bug

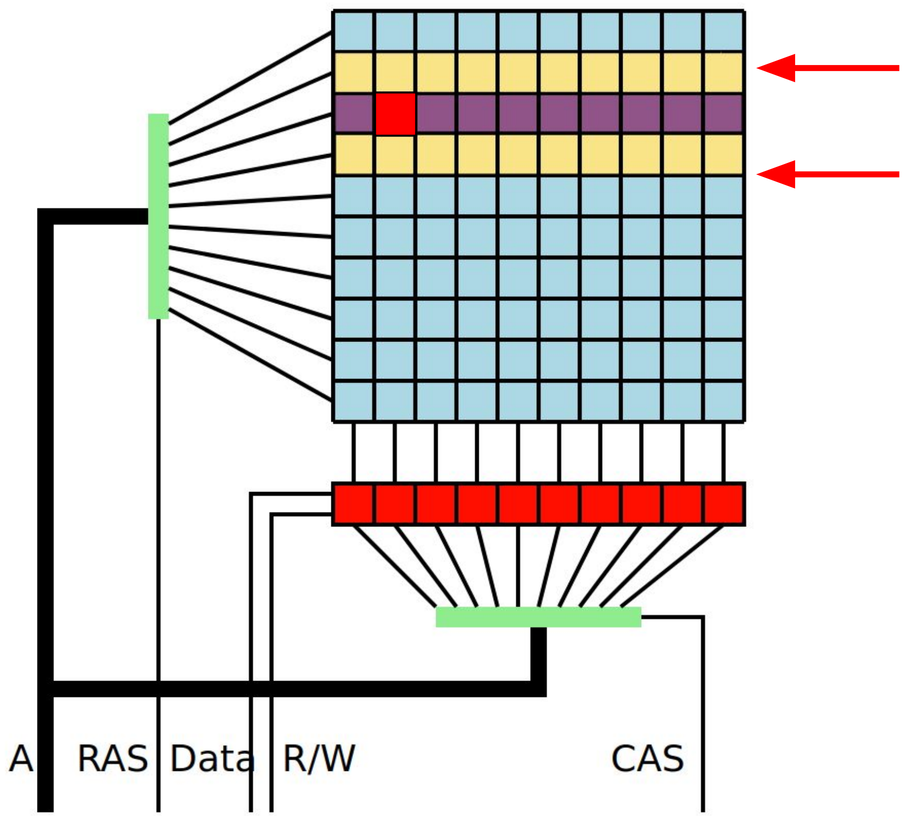


Image from wiki

Rowhammer bugs

- Rowhammer exploitation:
  - the attacker actively tries to cause bit flips...
  - ... in specific parts of the memory so to obtain an advantage...
  - ... which usually consists in getting root privileges
- Usual trick: push the OS to allocate a page table entry in a vulnerable location
  - Page table entries contain "virtual address ⇒ physical address" maps
  - If an attacker can tweak one, she can point a given VA to what she wants

Rowhammer bugs

Università degli Studi di Padova

- Rowhammer bugs have been exploited in many contexts and with many goals

- Examples
  - Escaping native client sandbox
  - Escaping javascript browser sandbox
  - Cross-VM exploitation
  - Drammer, rowhammer for ARM mobile platforms

"Traditional" bugs vs. design bugs

- Not all bugs are as easy to fix

- Traditional memory corruption bugs are "easy" to fix
  - Very well oiled pipeline to go from report to fix

- Design bugs are much more difficult to fix
  - Design bug: the design itself is broken, not just a small impl. detail
  - No standardized process: it's more difficult to report and get fixed
    - The fix may cause a significant code rewrite... (and devs don't like it)
    - ... which may introduce overhead / backward compatibility problems / new bugs

Questions? Feedback? Suggestions?