CSCI 445:
Mobile Application Security

Lecture 12

Prof. Adwait Nadkarni

Derived from Slides by William Enck

Announcements

- **Project Milestone 2 deadline on March 13th**
- **Use API level 32 or higher**
- **Update your Android Studio (allows Pixel 6 AVDs)**

Recap: Apps are programs

- What does a program do?: Transform inputs → outputs
- What are these inputs/outputs?
  - Network
  - Storage
  - User Interface
  - Sensors/Camera/Mic
  - Other applications
  - ...?

Program Vulnerabilities

Programming

- Why do we write programs?
  - Function
- What functions do we enable via our programs?
  - Some we want -- some we don't need
  - Adversaries take advantage of such "hidden" function

A Simple Program

```
int authenticated = 0;
char packet[1000];

while (!authenticated) {
  PacketRead(packet);
  if (Authenticate(packet))
    authenticated = 1;
}
 if (authenticated)
   ProcessPacket(packet);
```

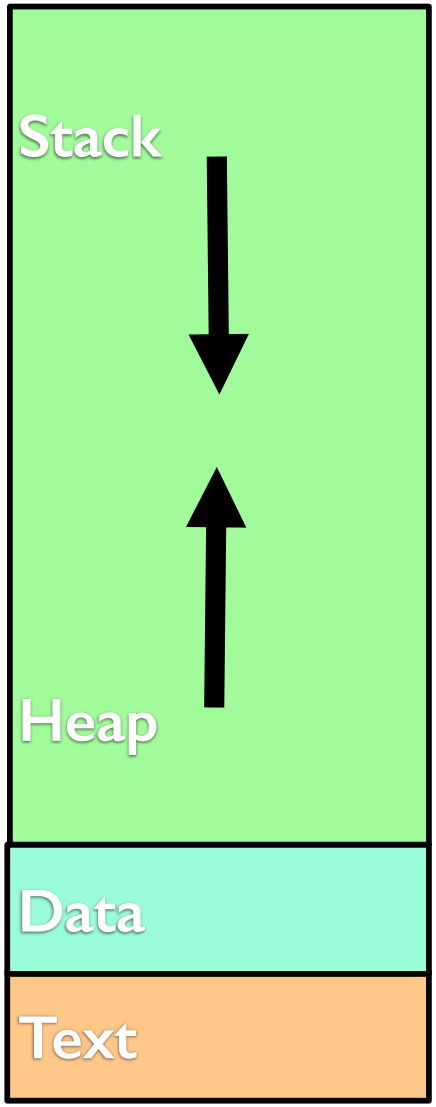
A Simple Program

```
int authenticated = 0;
char packet[1000];

while (!authenticated) {
  PacketRead(packet);
  if (Authenticate(packet))
    authenticated = 1;
}
 if (authenticated)
   ProcessPacket(packet);
```

What if packet is larger than 1000 bytes?

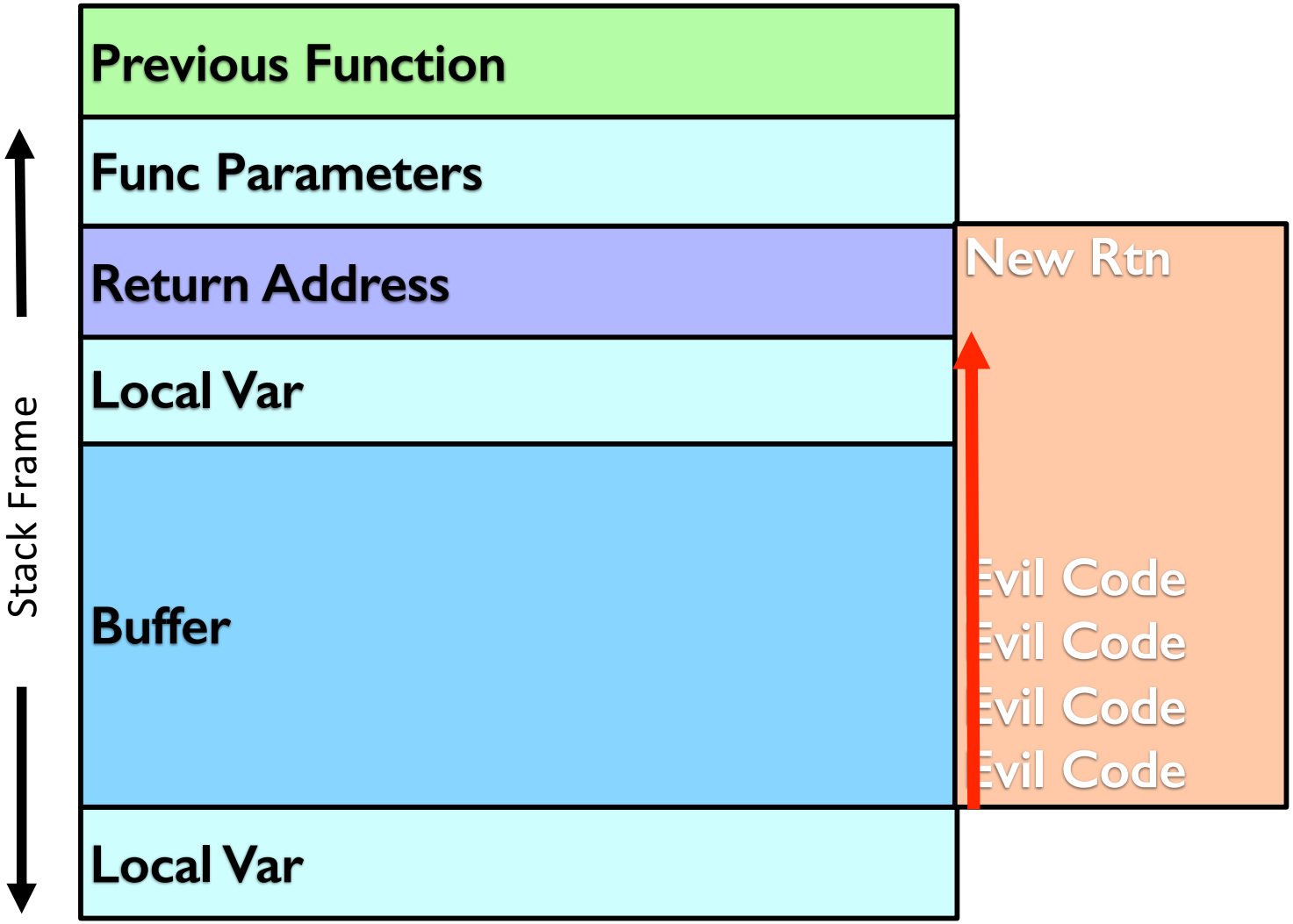
Address Space Layout



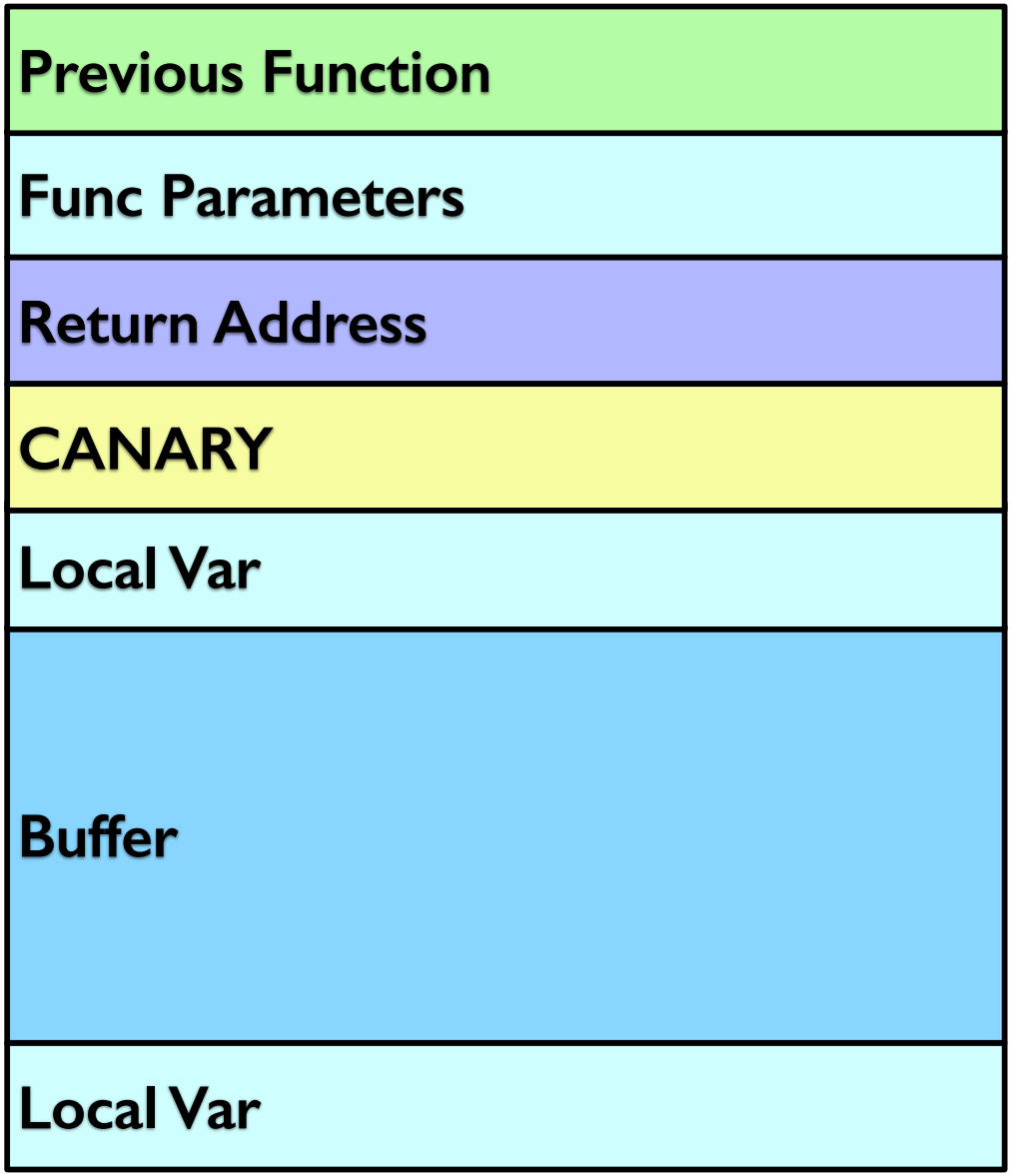
- Write beyond variable limit
  - Can write the without limits in some languages
- Can impact values
  - In heap, on stack, in data
- Can impact execution integrity
  - Can jump to arbitrary points in the program
    - Function pointers
    - Return addresses

Buffer Overflow

- How it works



Buffer Overflow Defense



- "Canary" on the stack
  - Random value placed between the local vars and the return address
  - If canary is modified, program is stopped
- Are we done?

A Simple Program

```
int authenticated = 0;
char packet[1000];

while (!authenticated) {
  PacketRead(packet);
   if (Authenticate(packet))
     authenticated = 1;
}
 if (authenticated)
   ProcessPacket(packet);
```

What if packet is only 1004 bytes?

Overflow of Local Variables

- Don't need to modify return address
  - Local variables may affect control
- What kinds of local variables would impact control?
  - Ones used in conditionals (example) and...?
  - Function pointers
- What can you do to prevent that?

A Simple Program

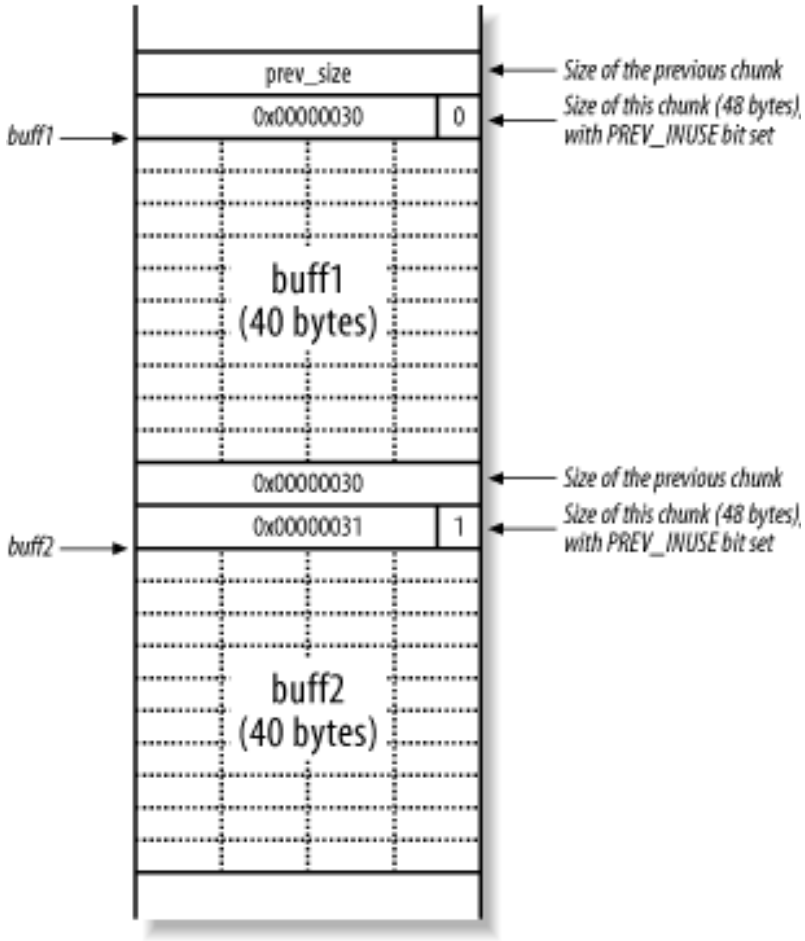
```
int authenticated = 0;
char *packet = (char *)malloc(1000);

while (!authenticated) {
  PacketRead(packet);
  if (Authenticate(packet))
    authenticated = 1;
}
 if (authenticated)
   ProcessPacket(packet);
```

What if we allocate the packet buffer on the heap?

Heap Overflow

- Overflows may occur on the heap also
  - Heap has data regions and metadata
- Attack
  - Write over heap with target address (heap spraying)
  - Hope that victim uses an overwritten function pointer before program crashes



Another Simple Program

```
int size = BASE_SIZE;
char *packet = (char *)malloc(1000);
char *buf = (char *)malloc(1000+BASE_SIZE);

 strcpy(buf, FILE_PREFIX);
 size += PacketRead(packet);
 if ( size < sizeof(buf)) {
   strcat(buf, packet);
   fd = open(buf);
 }
```

Any problem with this conditional check?

Integer Overflow

- Signed variables represent positive and negative values
  - Consider an 8-bit integer: -128 to 127
  - Weird math: *127+1 = ???*
- This results in some strange behaviors
  - **`size += PacketRead(packet)`**
    - What is the possible value of size?
  - **`if ( size < sizeof(buf)) {`**
    - What is the possible result of this
- How do we prevent these errors?

```
qsee_not_in_region(list, start,
start+size);
...
int qsee_not_in_region(void
*list, long start, long end)
{
        if (end < start)
        { tmp = start; start = end;
        end = tmp; }
        //     Perform validation ...
}
```

https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-On-Trusting-TrustZone-WP.pdf

A Simple Program

```
int authenticated = 0;
char *packet = (char *)malloc(1000);

while (!authenticated) {
  PacketRead(packet);
  if (Authenticate(packet))
    authenticated = 1;
}
if (authenticated)
   ProcessQuery("Select", partof(packet));
```

Any problem with this query request?

Parsing Errors

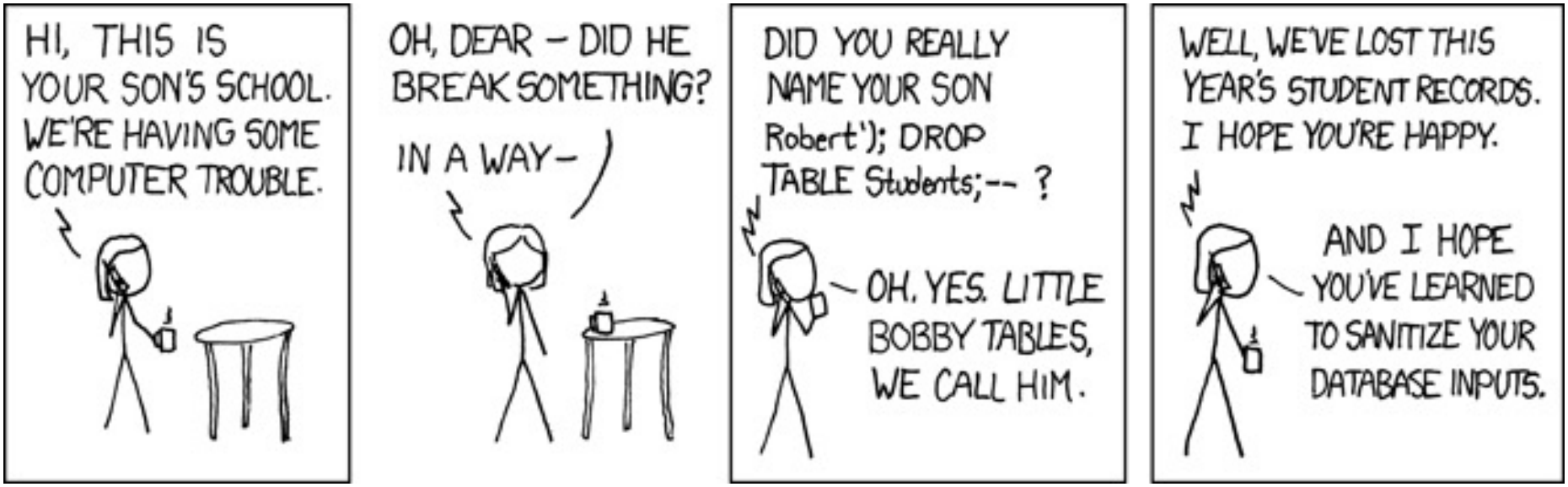
- Have to be sure that user input can only be used for expected function
  - *SQL injection*: user provides a substring for an SQL query that changes the query entirely (e.g., add SQL operations to query processing)

```
SELECT fieldlist FROM table
WHERE field = 'anything' OR 'x'='x';
```

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

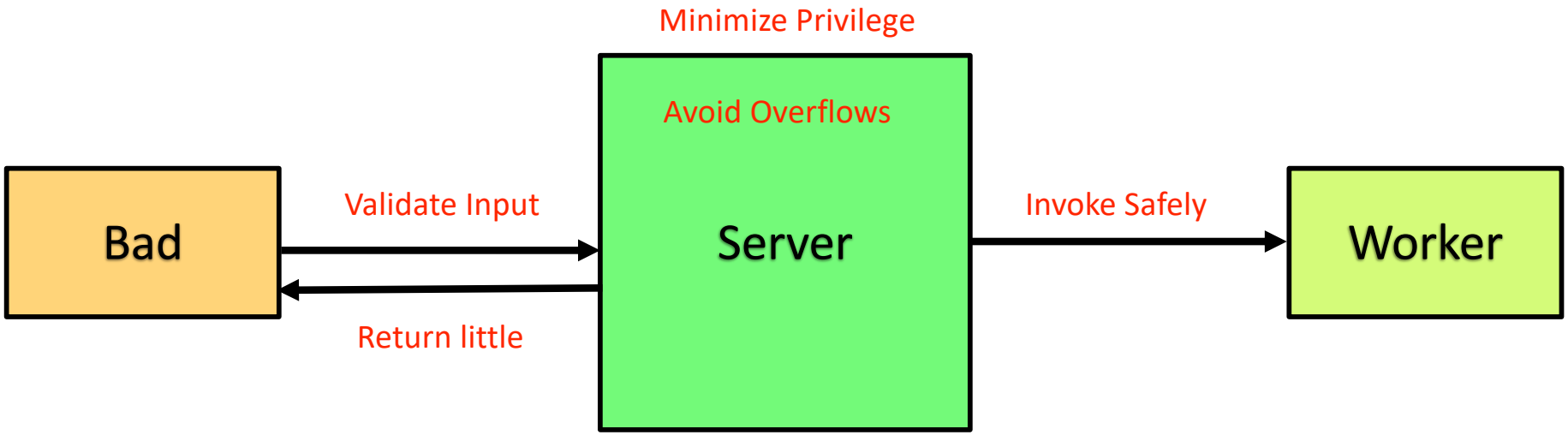
  - Goal: format all user input into expected types and ranges of values
    - Integers within range
    - Strings with expected punctuation, range of values

- Many scripting languages convert data between types automatically -- are not *type-safe* -- so must be extra careful


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Secure Input Handling

- David Wheeler's Secure Programming for Linux and UNIX
  - Validate all input; Only execute application-defined inputs!
  - Avoid the various overflows
  - Minimize process privileges
  - Carefully invoke other resources
  - Send information back carefully



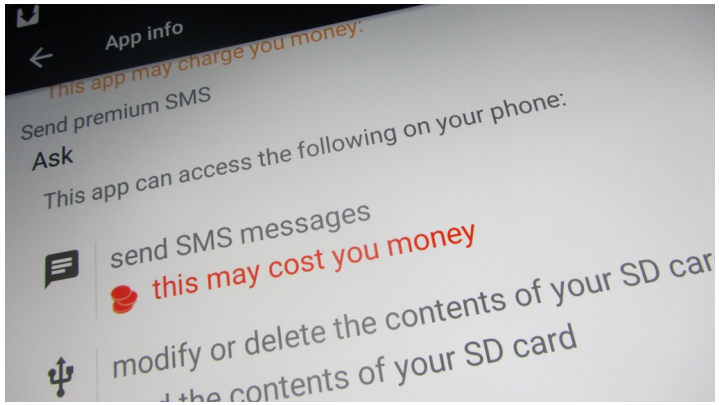
Application Analysis Goals

Why Study Apps?

- **Coarse Goals:** Find malware, bad behavior, understand what will break if we change things
- To elaborate:
  - Malicious behavior: What is *malicious?*
  - Vulnerable network communication
  - Privilege Escalation
  - Stealing private information
  - Permission misuse
  - Repackaging
  - Other potentially harmful behavior

Preventing Malware - I

- Like PC malware, smartphone malware is designed with an *incentive* in mind.
  - Usually boils down to making money
  - What does malware do?
    - Ransomware: Make important data unavailable
    - Premium-rate SMS
    - Mobile botnets
    - Spyware
    - Install backdoors, bring more malware...



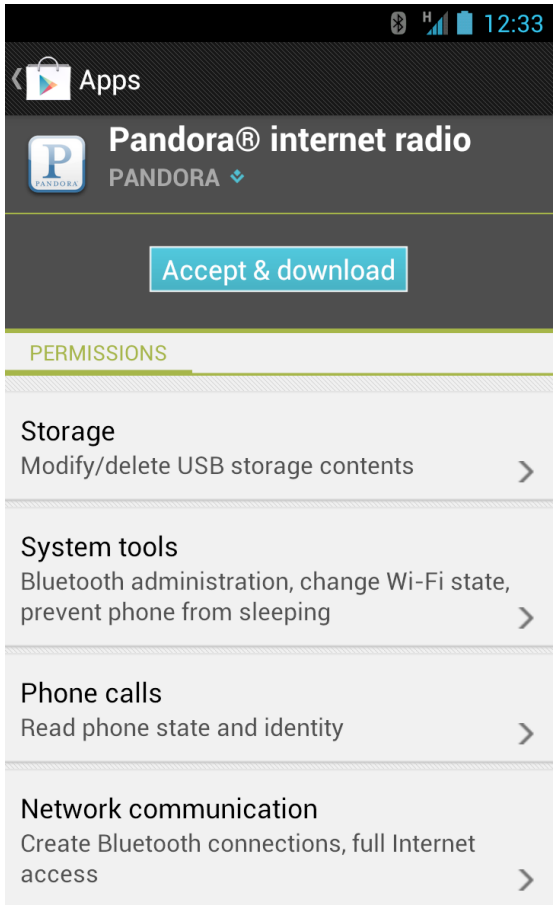
Preventing Malware - II

- Two types of malware:
  - Works within the permission system (most)
  - Acquires root-level access (harder to remove)
- Static and dynamic analysis in the market (Bouncer)
  - Inherent limitations: *what are you looking for?*
- *Class Exercise: Is on-phone antivirus software needed?*

lookout™
MOBILE SECURITY

Permission misuse

- Recap: *least privilege*
- Basic violation:
  - Ask for more permissions than you use
- More nuanced violation:
  - Ask for permissions that you use, but *shouldn't*
  - Why is this difficult to judge?
    - How do you decide what is appropriate?
    - Some ideas: based on UI, description, reviews, intuition, privacy policies



Stealing User Data

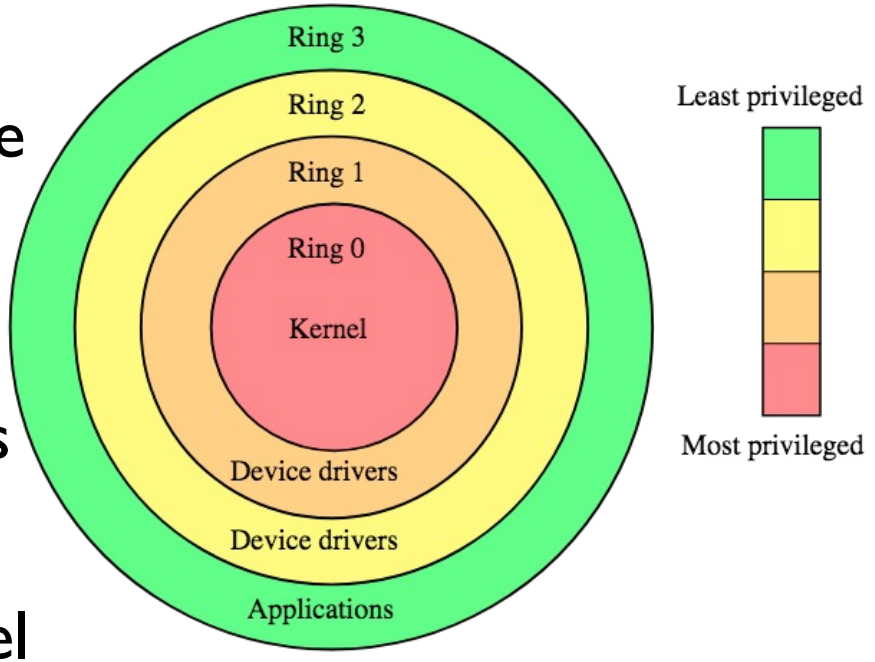
- What data are we talking about?:
  - Device data (OS controls access): device identifiers, location, contacts, calendar, photos
  - App-specific data (apps/user control access): Email, notes, files, etc.
- Q: Why do apps need user data?
- A: As a part of their functionality, to provide personalized service, advertising
- Goal: To find if apps are *stealing* private data
  - i.e., in the absence of user consent

SSL Vulnerabilities

- Apps are verifiers of SSL connections, but make mistakes
  - No certificate verification
  - No Hostname verification
  - ...
- Why is this bad?
  - Confidentiality: The adversary can steal your data
    - E.g., banking, shopping, social media
  - Integrity: The adversary can modify your data
    - E.g., banking, shopping, smart*
- *HW4*: *Automate* SSL misuse analysis

Privilege Escalation

- Even benign apps may have vulnerable interfaces
- If malware exploits such a vulnerable interface, what does that make the vulnerable app?
  - *Confused deputy*
- Sometimes, apps collude to combine privileges
- Other ways to escalate privilege:
  - Vulnerabilities in system services
  - Vulnerabilities in vendor apps
  - Vulnerabilities in the Linux kernel
  - ...



Repackaging

- Malware authors (1) download popular apps (2) disassemble them, (3) add malicious payload, and (4) distribute on official/unofficial app markets
  - Why would users install such apps?
    - *Free* versions of paid apps!
    - Identical to original app
    - Geographic constraints
- Detection at the market
- Still a problem. Why?
  - Available in unofficial markets
  - **Lesson:** Official markets only!

PHA (Potentially Harmful apps)

- **Grayware:** What makes it Gray?
  - Behavior that could be leveraged for a malicious objective,
  - but, we don't know that objective
- Examples:
  - Imposters: Impersonate popular apps
  - Madware: Aggressive ads (e.g., install shortcuts, change settings)
  - Misrepresentors (e.g., "weight scale", antivirus that does nothing)

Andow, Benjamin, Adwait Nadkarni, Blake Bassett, William Enck, and Tao Xie. "A study of grayware on google play." In *Security and Privacy Workshops (SPW), 2016 IEEE*, pp. 224-233. IEEE, 2016.

How do we study apps?

- Generally, two ways to do this:
- *Static analysis* tells you *what can potentially happen*
  - Getting source code: ded, dex2jar, jadx, androguard
  - Extending existing analysis frameworks (e.g., Fortify, soot)
  - Frameworks targeted at Android: FlowDroid, Argus
- *Dynamic analysis* tells you *what actually happened* in a specific runtime environment
  - Several tools: TaintDroid, DroidScope
  - Derivative environments: Droidbox, andrubis, MarvinSafe
  - *Hard to automate*; need to explore every code path in the app