RGBDroid: A Novel Response-based Approach to Android Privilege Escalation Attacks

Dankook University,

Massachusetts Institute of Technology, Konkuk University

2012, April 24th

Park Yeongung

santapark5 at gmail.com

Secure Software Lab.

KU KONKUK UNIVERSITY

DKU 단국대학교 DANKOOK UNIVERSITY

MIT Massachusetts Institute of Technology

What I will talk about..

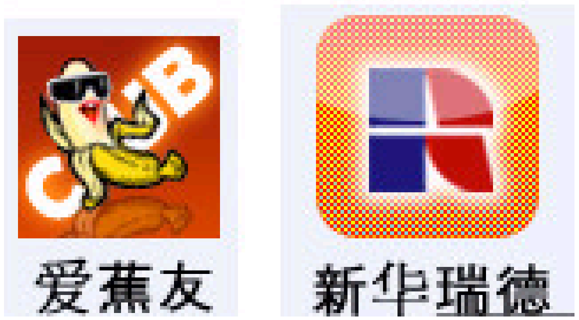
- Privilege escalation attack is dangerous especially on Android

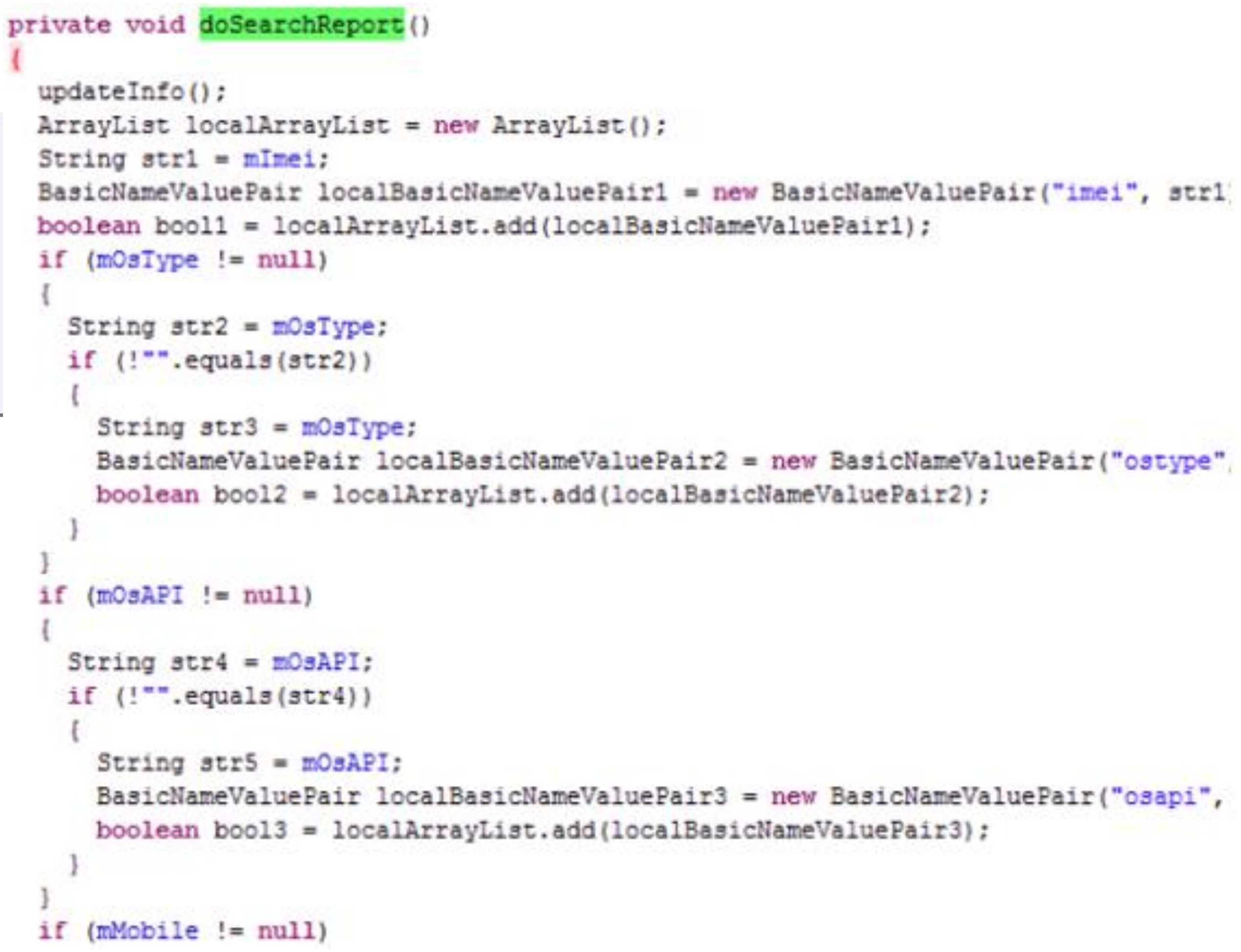
- Difference between prevention-oriented security and response-oriented security

- Since Android is a single user system and its native mechanism is static, we are able to predict its operations

Danger of privilege escalation attacks

- **DroidKungFu**





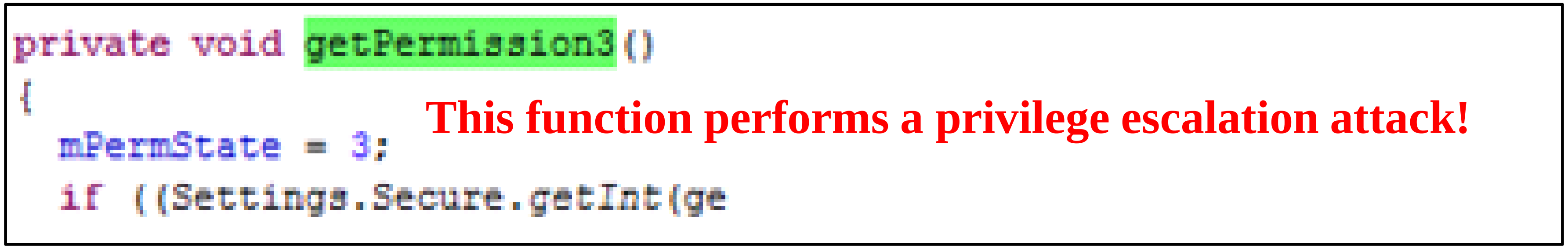
```
private void doSearchReport()
{
  updateInfo();
  ArrayList localArrayList = new ArrayList();
  String str1 = mImei;
  BasicNameValuePair localBasicNameValuePair1 = new BasicNameValuePair("imei", str1
  boolean bool1 = localArrayList.add(localBasicNameValuePair1);
  if (mOsType != null)
  {
    String str2 = mOsType;
    if (!"".equals(str2))
    {
      String str3 = mOsType;
      BasicNameValuePair localBasicNameValuePair2 = new BasicNameValuePair("ostype",
      boolean bool2 = localArrayList.add(localBasicNameValuePair2);
    }
  }
  if (mOsAPI != null)
  {
    String str4 = mOsAPI;
    if (!"".equals(str4))
    {
      String str5 = mOsAPI;
      BasicNameValuePair localBasicNameValuePair3 = new BasicNameValuePair("osapi",
      boolean bool3 = localArrayList.add(localBasicNameValuePair3);
    }
  }
  if (mMobile != null)
```

imei
ostype
osapi
model
SDKVersion
SDcard info
internal Memory Size
Net operator
phone number
running service

http://www.xinh*****.com:8111/GetCert/DevInfo?
http://search.go**********id.com:8511/search/getty.php
http://search.go**********id.com:8511/search/rpty.php

Danger of privilege escalation attacks

- **DroidKungFu**



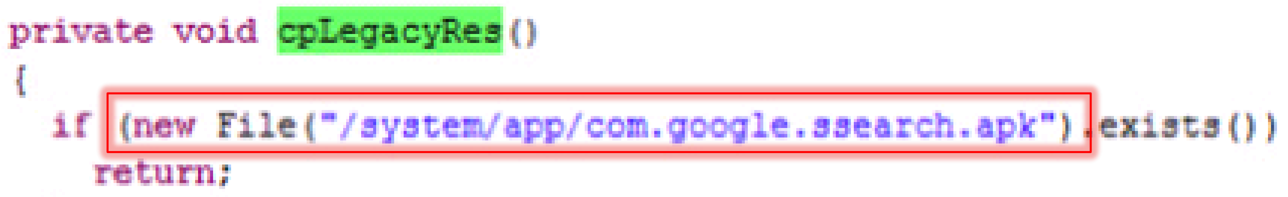
```
private void getPermission3()
{
  mPermState = 3;
  if ((Settings.Secure.getInt(ge
```

This function performs a privilege escalation attack!

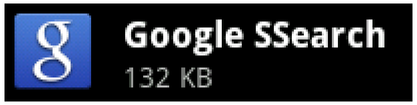
DroidKungFu is an embedded exploit code, which
is called ***"RageAgainstTheCage"*** and developed by C-SKILLS

After the privilege escalation attack!

DroidKungFu installs additional malicious app in ***'asset'*** directory

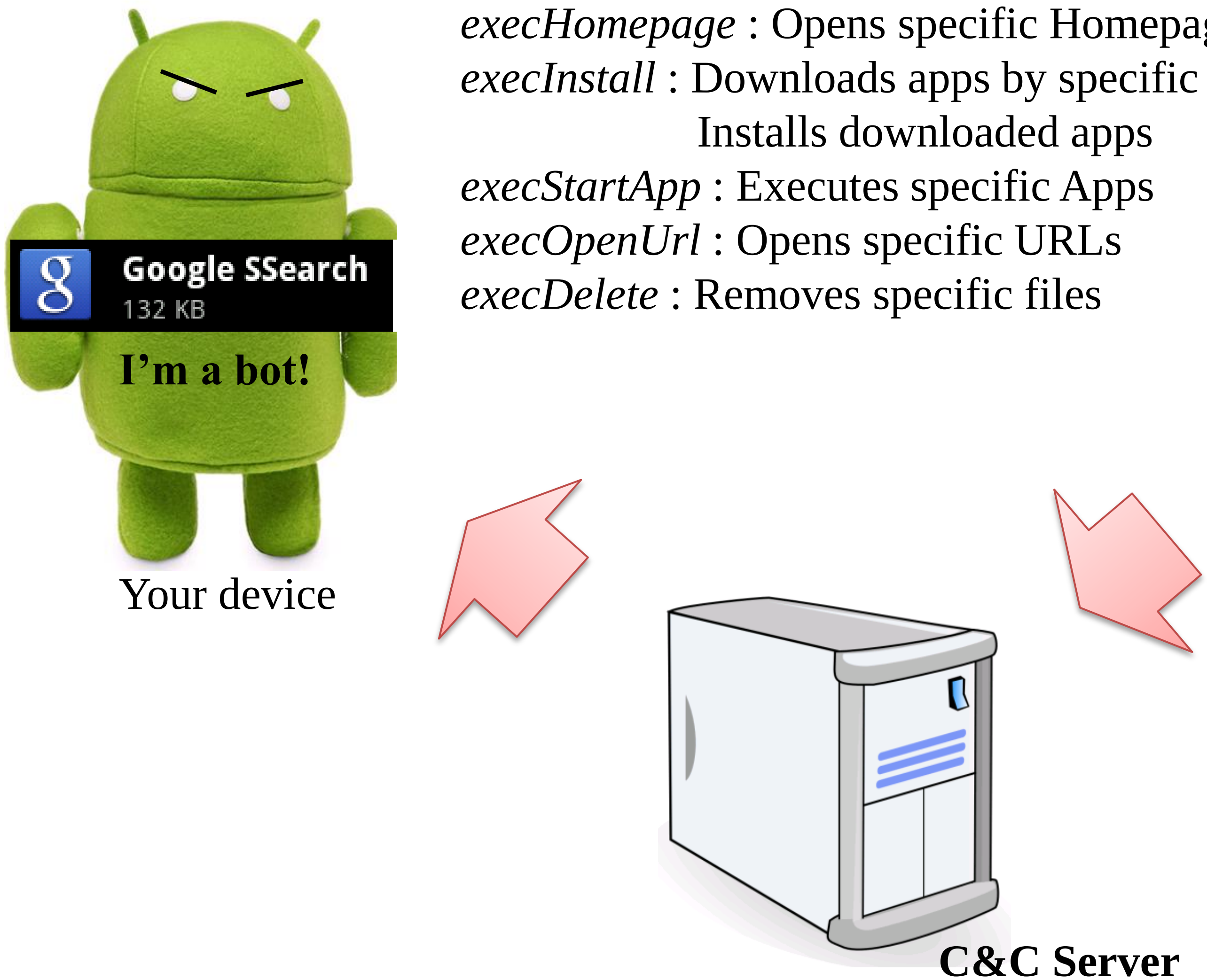


```
private void cpLegacyRes()
{
  if (new File("/system/app/com.google.ssearch.apk").exists())
    return;
```



Danger of privilege escalation attacks

- DroidKungFu

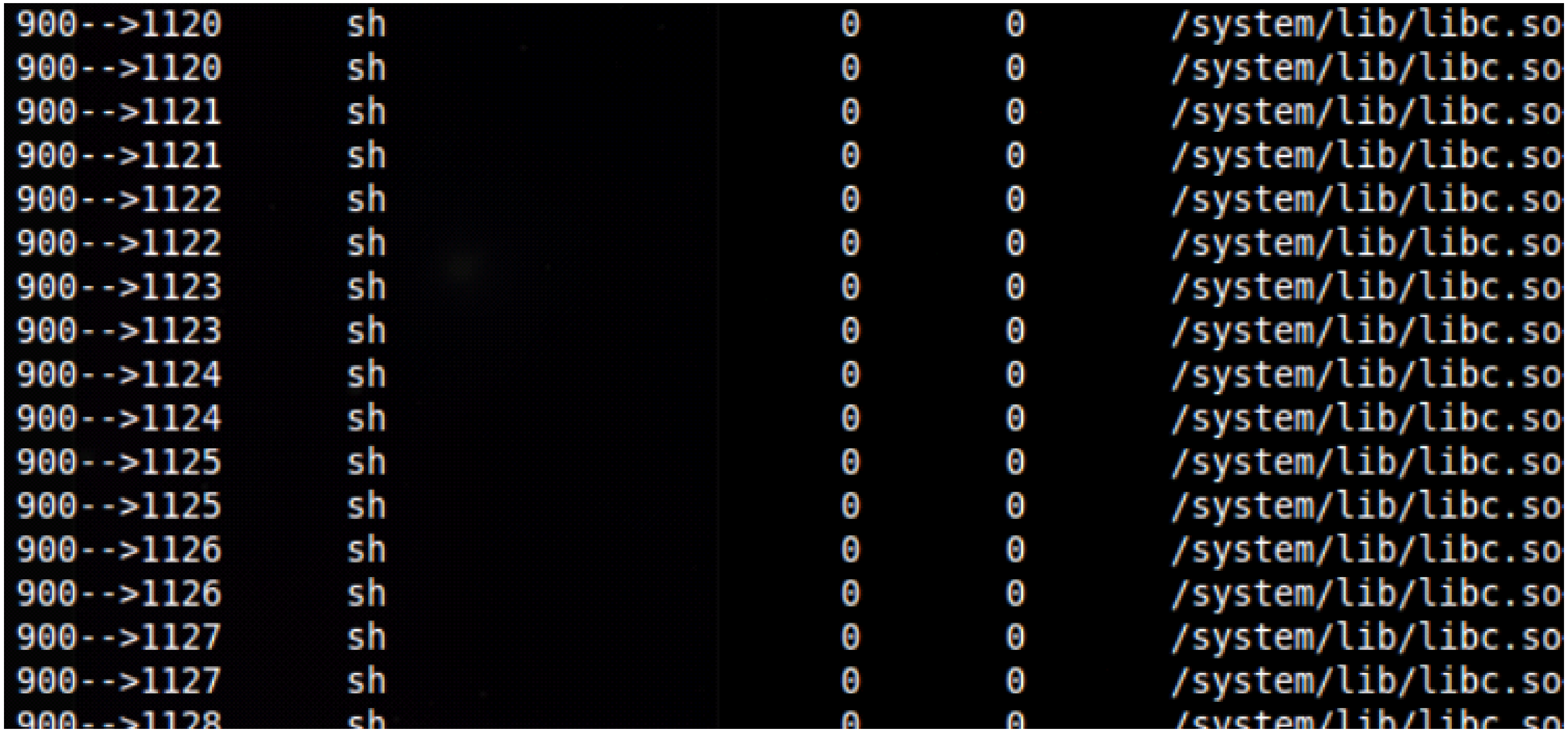


execHomepage : Opens specific Homepages
execInstall : Downloads apps by specific URLs, Installs downloaded apps
execStartApp : Executes specific Apps
execOpenUrl : Opens specific URLs
execDelete : Removes specific files

Android works statically and predictably

- Analyzed file access patterns hooking system calls in Android
- Also identified processes which run with root privileges

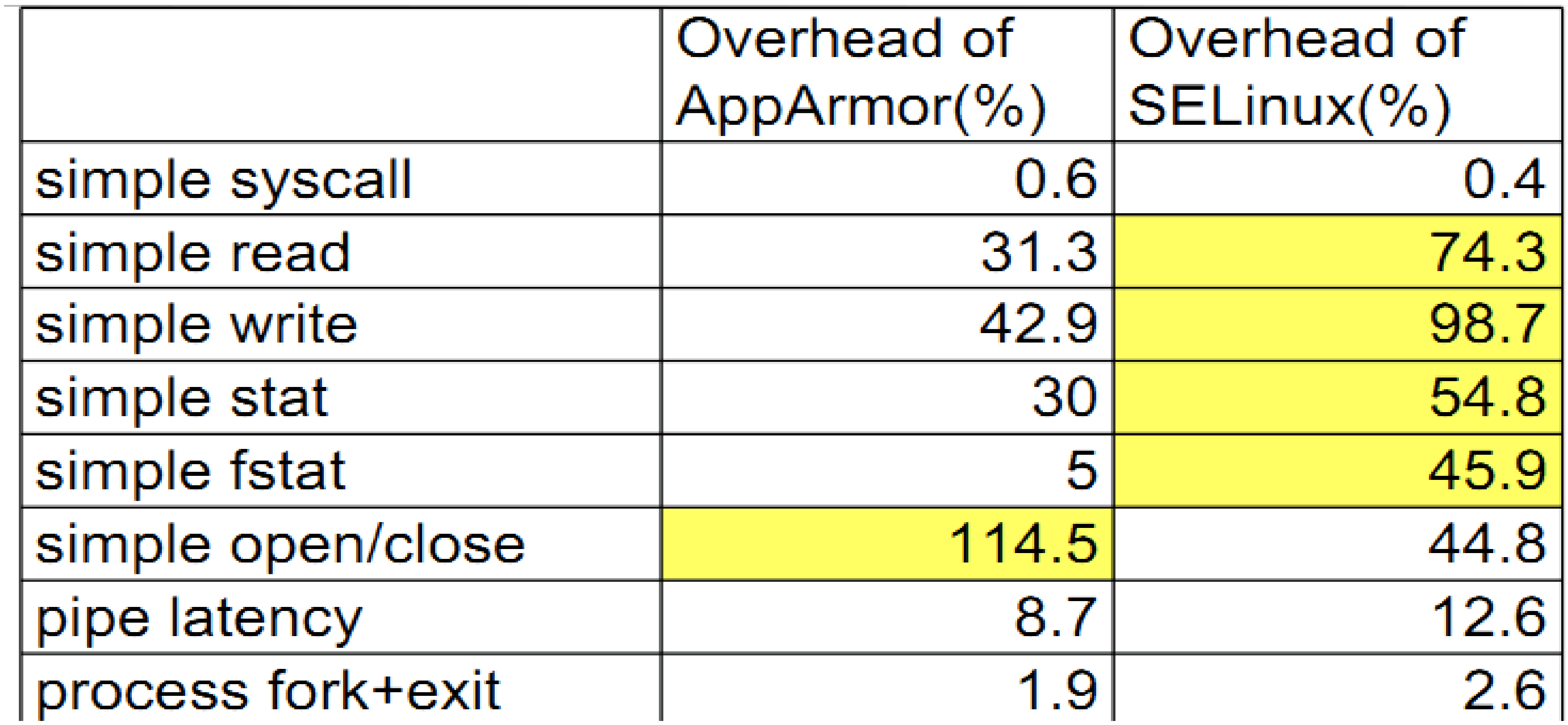
ppid→pid process name uid euid file to access



```
900-->1120      sh          0       0       /system/lib/libc.so
900-->1120      sh          0       0       /system/lib/libc.so
900-->1121      sh          0       0       /system/lib/libc.so
900-->1121      sh          0       0       /system/lib/libc.so
900-->1122      sh          0       0       /system/lib/libc.so
900-->1122      sh          0       0       /system/lib/libc.so
900-->1123      sh          0       0       /system/lib/libc.so
900-->1123      sh          0       0       /system/lib/libc.so
900-->1124      sh          0       0       /system/lib/libc.so
900-->1124      sh          0       0       /system/lib/libc.so
900-->1125      sh          0       0       /system/lib/libc.so
900-->1125      sh          0       0       /system/lib/libc.so
900-->1126      sh          0       0       /system/lib/libc.so
900-->1126      sh          0       0       /system/lib/libc.so
900-->1127      sh          0       0       /system/lib/libc.so
900-->1127      sh          0       0       /system/lib/libc.so
900-->1128      sh          0       0       /system/lib/libc.so
```

Prevention vs. Response

- **Prevention-oriented security may cause high overhead**



	Overhead of AppArmor(%)	Overhead of SELinux(%)
simple syscall	0.6	0.4
simple read	31.3	74.3
simple write	42.9	98.7
simple stat	30	54.8
simple fstat	5	45.9
simple open/close	114.5	44.8
pipe latency	8.7	12.6
process fork+exit	1.9	2.6

Prevention vs. Response

- Prevention-oriented security solutions must predict all potential attacks and vulnerabilities
  - To do so, the overall threat and risk analysis is required
  - This can cause high overhead → It is almost impossible
  - Moreover, these solutions may not explicitly describe what they prevent.
  - Therefore, these solutions are not perfect

Prevention vs. Response

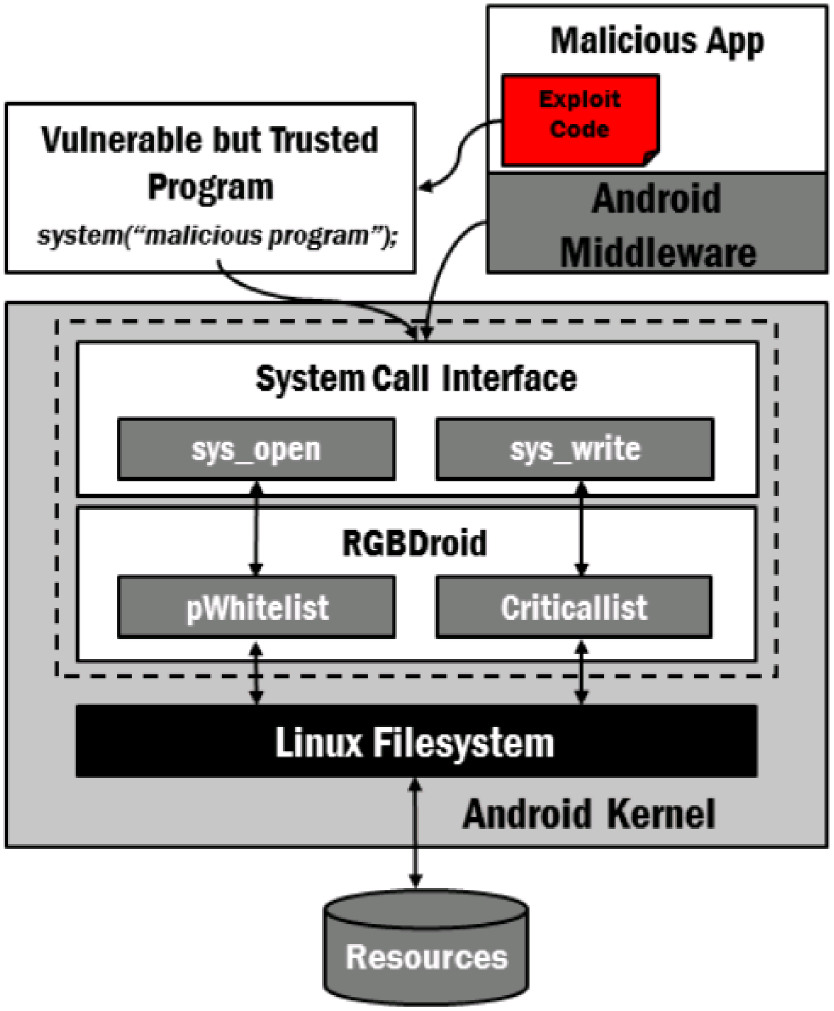
- Our response-oriented security first defines critical malicious behaviors to be potential dangers under the assumption that Android system was compromised by attacker

- We then make a response policy for each defined malicious behavior considering features of the Android system
  - We apply this response policy to our security approach

- We have designed and implemented *RGBDroid* system for the response-oriented security approach

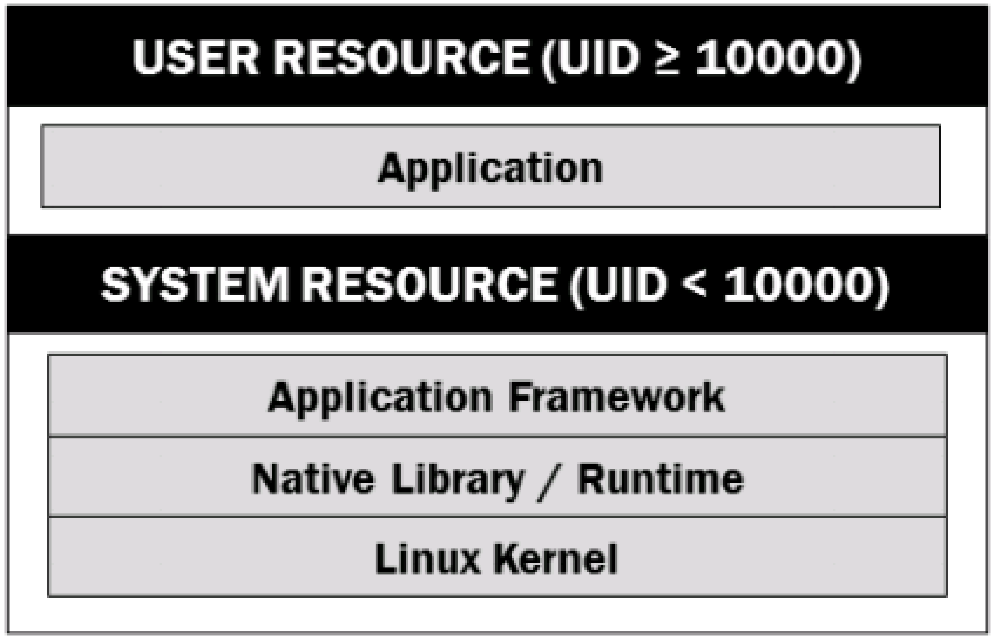
RGBDroid overview

- Android statically works with following the standard policy
  - The root privileges are used only by specific processes
  - There are critical system resources which can be modified by a designated process



RGBDroid overview

- User layer resources are owned by the accounts whose UID is greater than or equal to 10000
- System layer resources are owned by the accounts whose UID is less than 10000



pWhitelist in RGBDroid

- pWhitelist is the list of programs that can run with root privileges

- Root privilege in Android is only used by specific processes (ex. daemons)

- RGBDroid denies any resources access request made by a program which is not a member of pWhitelist

```
unsigned short uid;
unsigned short euid;

if uid == 0 OR euid == 0
    if !(procname == procname_in_whiltelist)
        return deny;
call sys_open();
```

Criticallist in RGBDroid

- **Criticallist is a list of system layer resources that even a process with root privilege cannot modify.**

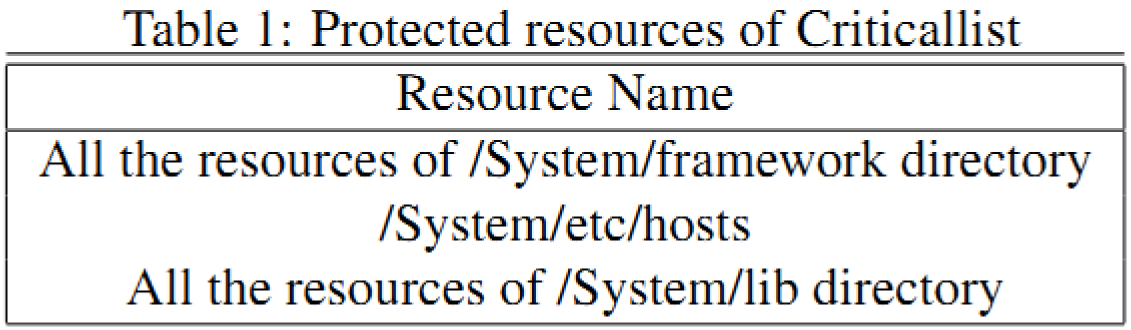
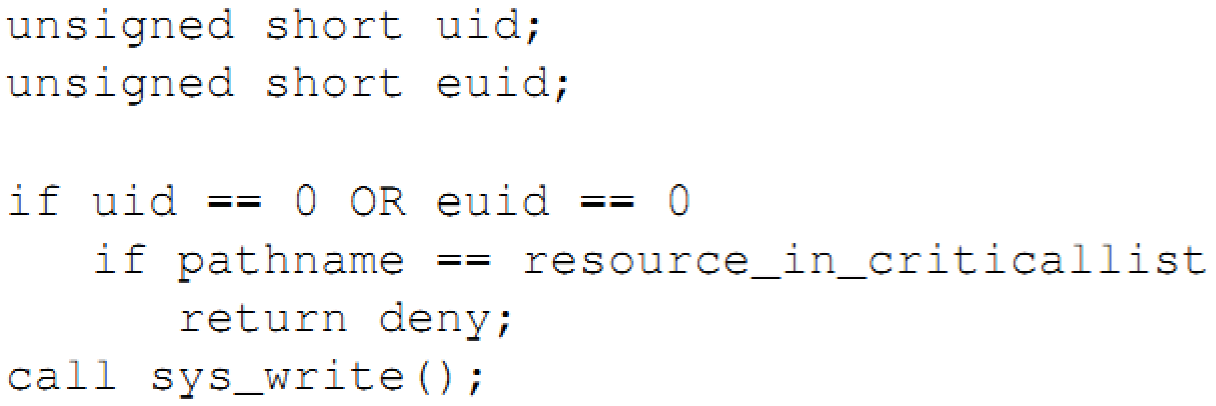


Table 1: Protected resources of Criticallist

Resource Name
All the resources of /System/framework directory
/System/etc/hosts
All the resources of /System/lib directory

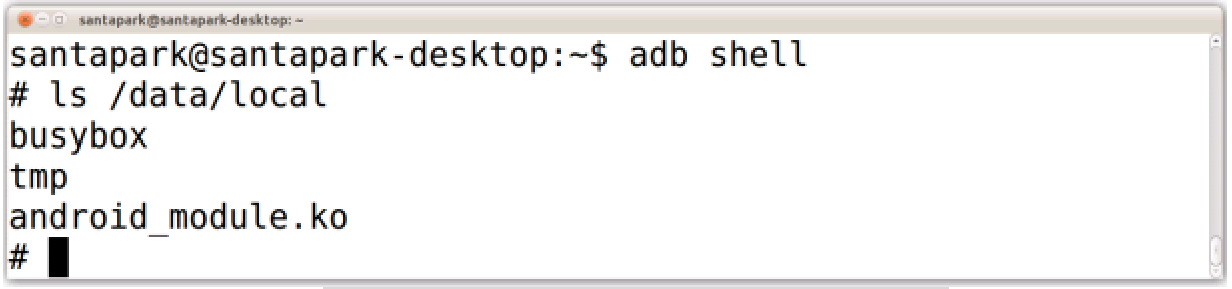


```
unsigned short uid;
unsigned short euid;

if uid == 0 OR euid == 0
    if pathname == resource_in_criticallist
        return deny;
call sys_write();
```

What we can response..

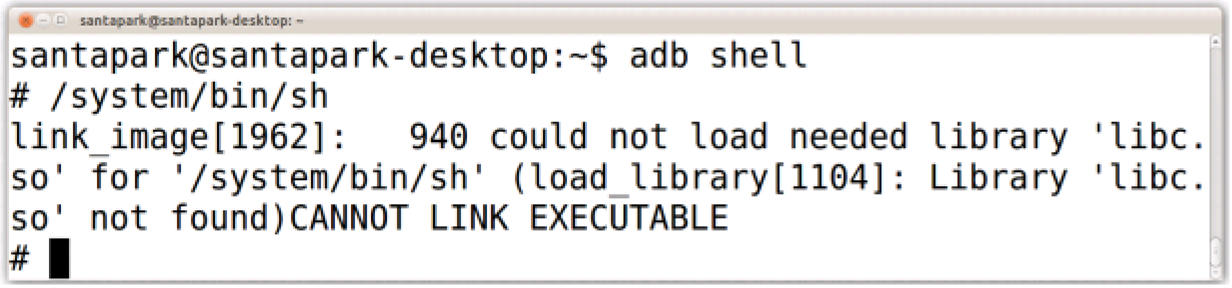
- Shell acquisition: Many attacks try to get a root shell
- pWhitelist in RGBDroid prevents illegal access to the root shell and disallows the attempt



santapark@santapark-desktop: ~

```
santapark@santapark-desktop:~$ adb shell
# ls /data/local
busybox
tmp
android_module.ko
# 
```

After apply RGBDroid



santapark@santapark-desktop: ~

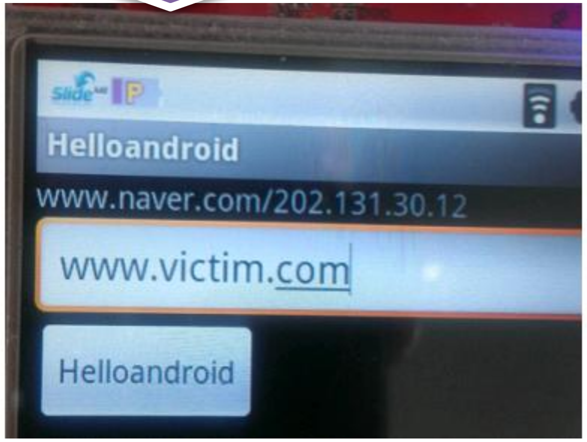
```
santapark@santapark-desktop:~$ adb shell
# /system/bin/sh
link_image[1962]:   940 could not load needed library 'libc.
so' for '/system/bin/sh' (load_library[1104]: Library 'libc.
so' not found)CANNOT LINK EXECUTABLE
# 
```

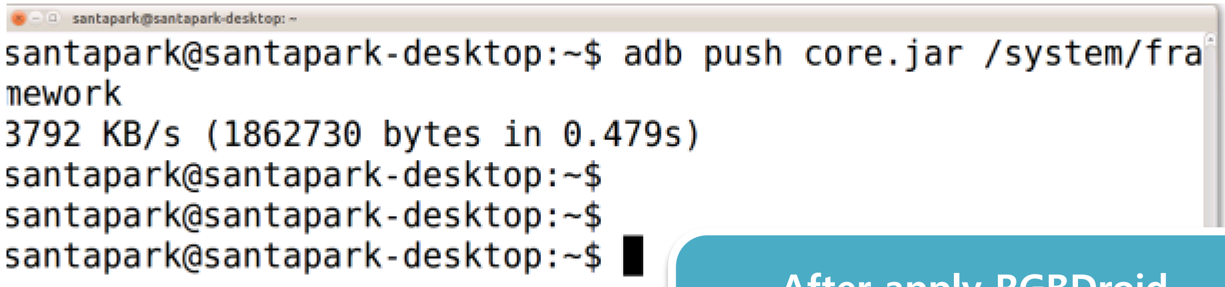
What we can response..

- Restrict illegal modification of critical system resources
- Attacker can do various malicious things by manipulating the resources

  (ex. /system/framework/core.jar, framework.jar, hosts, etc.)

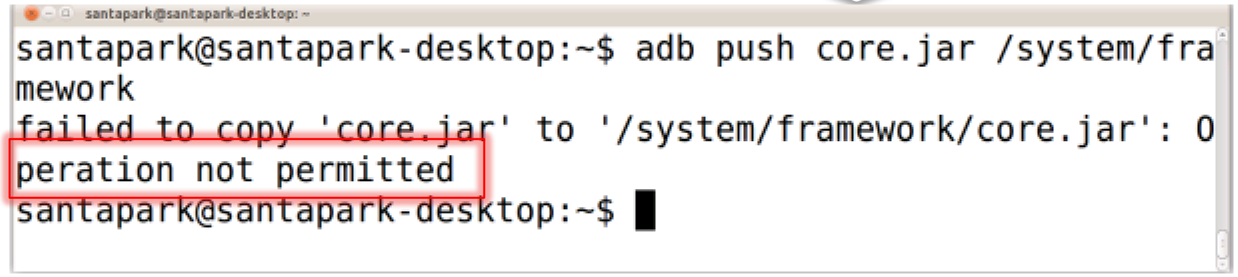
DNS Spoofing:
Request: www.victim.com
Redirection : www.naver.com





```
santapark@santapark-desktop:~$ adb push core.jar /system/fra
mework
3792 KB/s (1862730 bytes in 0.479s)
santapark@santapark-desktop:~$
santapark@santapark-desktop:~$
santapark@santapark-desktop:~$
```

After apply RGBDroid
Manipulation of critical system resource will fail ☺



```
santapark@santapark-desktop:~$ adb push core.jar /system/fra
mework
failed to copy 'core.jar' to '/system/framework/core.jar': O
peration not permitted
santapark@santapark-desktop:~$
```

Performance Evaluation

- After applying RGBDroid, I/O throughput diminishes by 6.2%, 6.7%, 8.1% for insertion, update, and deletion operation respectively
- The overall average I/O throughput decreases by 7%

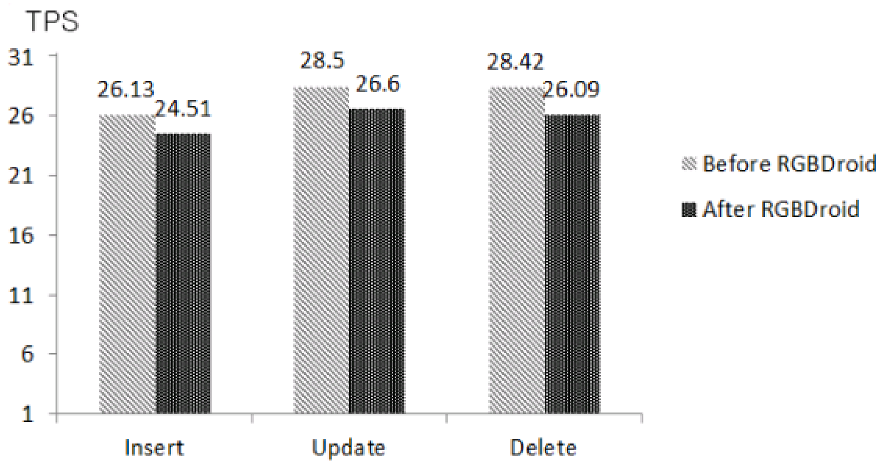


Table 2: I/O Performance Measurement Table (Unit: TPS (Transactions Per Second))

Count	Before RGBDroid			After RGBDroid		
	Insert	Update	Delete	Insert	Update	Delete
1	25.77	28.17	28.28	24.83	26.56	26.71
2	26.02	28.69	28.1	25.22	26.83	26.67
3	26.14	28.95	28.58	24.84	27.17	23.95
4	26.8	28.72	28.76	23.95	26.36	26.69
5	25.94	28.81	28.3	22.98	26.23	25.36
6	27.4	28.4	28.79	24.78	25.52	26.44
7	24.51	28.67	28.66	23.25	26.69	26.03
8	27.23	27.37	28.5	25.09	27.23	26.89
9	24.49	28.53	27.55	25.03	26.1	26.5
10	26.99	28.73	28.67	25.12	27.33	25.64
Ave.	26.13	28.50	28.42	24.51	26.60	26.09

Performance Evaluation

- Processing time increases by 6.2%, 6.7%, and 8.4% for each operation after RGBDroid is applied.
- Average processing time for all three operations increases by 7% overall, which can be considered small processing overhead

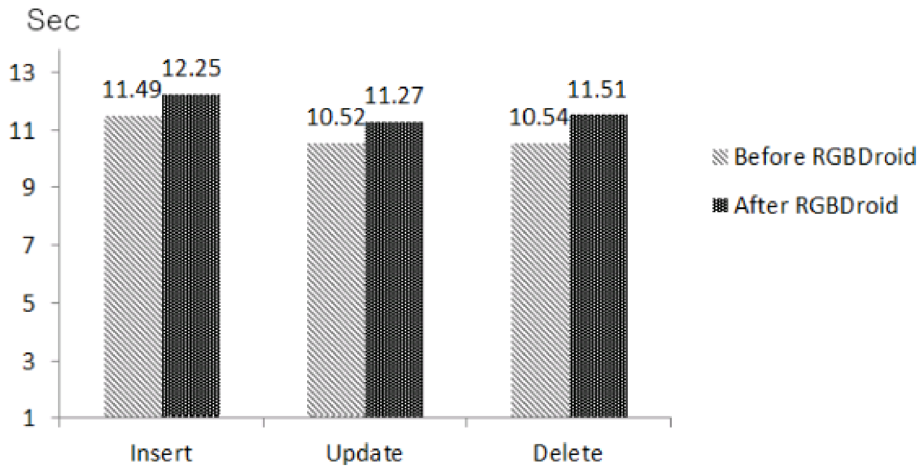


Table 3: User processing time measurement table (Unit: second)

Count	Before RGBDroid			After RGBDroid		
	Insert	Update	Delete	Insert	Update	Delete
1	11.64	10.64	10.6	12.07	11.29	11.23
2	11.52	10.45	10.67	11.89	11.17	11.24
3	11.47	10.36	10.36	12.07	11.04	12.52
4	11.19	10.44	10.42	12.52	11.37	11.23
5	11.56	10.41	10.59	13.05	11.43	11.82
6	10.94	10.56	10.42	12.1	11.75	11.34
7	12.23	10.46	10.46	12.9	11.23	11.52
8	11.01	10.95	10.52	11.95	11.01	11.16
9	12.24	10.51	10.88	11.98	11.49	11.31
10	11.11	10.44	10.46	11.94	10.97	11.7
Ave.	11.49	10.52	10.54	12.25	11.27	11.51

Analysis of Our Approach

- Predicting all possible vulnerabilities is unrealistic both in principle as well as in practice.
- Response-based approach does not have to consider how vulnerabilities can be exploited
- Response-based approach also explicitly specify what the security system responses
  - It does not need to monitor and trace all accesses to critical resources.
  - It does not require monitoring numerous parts of the system *(does need a few additional operations*)
  - It causes only a small performance overhead unlike the prevention approach.

Conclusion

- In the Android, recent malware illegally manipulates system resources or turns the system into a bot by privilege escalation attacks
- This paper presented RGBDroid system for response-based security approach
  - It does not require monitoring or predicting all the potential vulnerabilities but just requires blocking possible malicious acts after attacks
  - It is very suitable for Android environment
- We have plan to evolve our response-based security approach into malicious behavior-oriented security one

Any questions?

THANK YOU FOR YOUR ATTENTION!