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.
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

```java
private void doSearchReport()
{
  updateInfo();
  ArrayList localArrayList = new ArrayList();
  String str1 = imei;
  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", str3);
      boolean bool2 = localArrayList.add(localBasicNameValuePair2);
    }
  }
  if (mOsAPI != null)
  {
    String str4 = mOsAPI;
    if (!"".equals(str4))
    {
      String str5 = mOsAPI;
      BasicNameValuePair localBasicNameValuePair3 = new BasicNameValuePair("osapi", str5);
      boolean bool3 = localArrayList.add(localBasicNameValuePair3);
    }
  }
  if (mMobile != null)
  {
    String str6 = mMobile;
    if (!"".equals(str6))
    {
      String str7 = mMobile;
      BasicNameValuePair localBasicNameValuePair4 = new BasicNameValuePair("sdc", str7);
      boolean bool4 = localArrayList.add(localBasicNameValuePair4);
    }
  }
}
```

- 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**

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

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

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

Google SSearch
132 KB
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

Your device → C&C Server → attacker

I’m a bot!
Android works statically and predictably

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

<table>
<thead>
<tr>
<th>ppid→pid</th>
<th>process name</th>
<th>uid</th>
<th>euid</th>
<th>file to access</th>
</tr>
</thead>
<tbody>
<tr>
<td>900→1120</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1120</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1121</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1121</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1122</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1122</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1123</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1123</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1124</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1124</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1125</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1125</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1126</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1126</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1127</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1127</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
<tr>
<td>900→1128</td>
<td>sh</td>
<td>0</td>
<td>0</td>
<td>/system/lib/libc.so</td>
</tr>
</tbody>
</table>
Prevention vs. Response

- Prevention-oriented security may cause high overhead

<table>
<thead>
<tr>
<th>Operation</th>
<th>Overhead of AppArmor(%)</th>
<th>Overhead of SELinux(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple syscall</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>simple read</td>
<td>31.3</td>
<td>74.3</td>
</tr>
<tr>
<td>simple write</td>
<td>42.9</td>
<td>98.7</td>
</tr>
<tr>
<td>simple stat</td>
<td>30</td>
<td>54.8</td>
</tr>
<tr>
<td>simple fstat</td>
<td>5</td>
<td>45.9</td>
</tr>
<tr>
<td>simple open/close</td>
<td>114.5</td>
<td>44.8</td>
</tr>
<tr>
<td>pipe latency</td>
<td>8.7</td>
<td>12.6</td>
</tr>
<tr>
<td>process fork+exit</td>
<td>1.9</td>
<td>2.6</td>
</tr>
</tbody>
</table>
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
• 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

```c
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

<table>
<thead>
<tr>
<th>Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>All the resources of /System/framework directory</td>
</tr>
<tr>
<td>/System/etc/hosts</td>
</tr>
<tr>
<td>All the resources of /System/lib directory</td>
</tr>
</tbody>
</table>

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
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

After apply RGBDroid
Manipulation of critical system resource will fail 😊
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%

![Chart showing I/O performance comparison before and after RGBDroid]

<table>
<thead>
<tr>
<th>Count</th>
<th>Before RGBDroid</th>
<th>After RGBDroid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Insert</td>
<td>Update</td>
</tr>
<tr>
<td>1</td>
<td>25.77</td>
<td>28.17</td>
</tr>
<tr>
<td>3</td>
<td>26.14</td>
<td>28.95</td>
</tr>
<tr>
<td>4</td>
<td>26.8</td>
<td>28.72</td>
</tr>
<tr>
<td>5</td>
<td>25.94</td>
<td>28.81</td>
</tr>
<tr>
<td>6</td>
<td>27.4</td>
<td>28.4</td>
</tr>
<tr>
<td>7</td>
<td>24.51</td>
<td>28.67</td>
</tr>
<tr>
<td>8</td>
<td>27.23</td>
<td>27.37</td>
</tr>
<tr>
<td>9</td>
<td>24.49</td>
<td>28.53</td>
</tr>
<tr>
<td>10</td>
<td>26.99</td>
<td>28.73</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Count</th>
<th>Before RGBDroid</th>
<th>After RGBDroid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Insert</td>
<td>Update</td>
</tr>
<tr>
<td>1</td>
<td>11.64</td>
<td>10.64</td>
</tr>
<tr>
<td>2</td>
<td>11.52</td>
<td>10.45</td>
</tr>
<tr>
<td>3</td>
<td>11.47</td>
<td>10.36</td>
</tr>
<tr>
<td>4</td>
<td>11.19</td>
<td>10.44</td>
</tr>
<tr>
<td>5</td>
<td>11.56</td>
<td>10.41</td>
</tr>
<tr>
<td>6</td>
<td>10.94</td>
<td>10.56</td>
</tr>
<tr>
<td>7</td>
<td>12.23</td>
<td>10.46</td>
</tr>
<tr>
<td>8</td>
<td>11.01</td>
<td>10.95</td>
</tr>
<tr>
<td>9</td>
<td>12.24</td>
<td>10.51</td>
</tr>
<tr>
<td>10</td>
<td>11.11</td>
<td>10.44</td>
</tr>
</tbody>
</table>

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!