One Class to Rule Them All

USENIX WOOT ’15

Or Peles & Roee Hay
IBM Security
We will see how this Android SDK class

```java
public class OpenSSLX509Certificate extends X509Certificate {

    private final long mContext;

    ...

}
```

MISSING MODIFIER BEFORE OUR DISCLOSURE!
(NOW PATCHED)
Led to this...

- REPLACEMENT OF APPS
- SELINUX BYPASS
- ACCESS TO APPS’ DATA
- KERNEL CODE EXECUTION*

* On select devices

One Class to Rule Them All (Or Peles & Roee Hay, USENIX WOOT ’15)
Introduction
Android Inter-App Communication 101
Android Inter-App Communication 101

Intent { play:// …}
Android Inter-App Communication 101

Intent { play:// ...} →

Intent { sms:// ...} →
An Intent can also contain

Bundle

SIMPLE OBJECTS
An Intent can also contain

**Bundle**

- **SIMPLE OBJECTS**
- **SERIALIZABLE OBJECTS**
Motivation
CVE-2014-7911 (Jann Horn):

Non-Serializable Classes can be Deserialized on target.
Exploiting CVE-2014-7911

Step 1. Find an interesting target.

MALWARE

TARGET
system_server
Exploiting CVE-2014-7911

Step 2. Send target a ‘serialized’ object in a Bundle

MALWARE

SYSTEM_SERVER

Serialized Object of non-Serializable class
final class BinderProxy implements IBinder {
    private long mOrgue; /* CONTROLALBLE NATIVE OBJECT */

    ...

    private native final destroy(); /* USES THE OBJECT */

    @Override
    protected void finalize() throws Throwable {
        try {
            destroy();
        } finally {
            super.finalize();
        }
    }
}
Exploiting CVE-2014-7911

Step 3. Make it deserialize on the target.

MALWARE

SYSTEM_SERVER

Deserialized Object
Make it deserialize automatically

All Bundle members are deserialized with a single ‘touch’ on the incoming Bundle:

E.g.

```java
public String getString(String key) {
    unparsed(); ← DESERIALIZES ALL
    final Object o = mMap.get(key);
    try {
        return (String) o;
    } catch (ClassCastException e) {...}
}
```
Exploiting CVE-2014-7911

Step 4. Make one of its methods *execute* on target.
(4) Make it Execute some Sensitive Code

```java
final class BinderProxy implements IBinder {
    private long mOrgue;

    ...

    private native final destroy();

    @Override
    protected void finalize() throws Throwable {
        try {
            destroy();
        }
        finally {
            super.finalize();
        }
    }
}
```

EXECUTED AUTOMATICALLY BY THE GC
Google’s Fix for CVE-2014-7911

Do Not Deserialize Non-Serializable Classes
Our 1st Contribution:
The Android Vulnerability

CVE-2015-3825
Our Research Question: A Potential Vulnerability

class Foo implements Serializable {
    private long mObject;

    // Pointer used in native code.
    private native final destroy();

    @Override
    protected void finalize() throws Throwable {
        try {
            destroy();
        } finally {
            super.finalize();
        }
    }
}
Experiment 1

boot.art
Experiment 1

boot.art

App: Loaded classes using Reflection

~13K Loadable Java Classes
Experiment 1

boot.art

~13K Loadable Java Classes

App: Loaded classes using Reflection

Dumped classes:
1. Serializable
2. Finalize method
3. Controllable fields
OpenSSLX509Certificate
public class OpenSSLX509Certificate extends X509Certificate {

    private final long mContext;

    @Override
    protected void finalize() throws Throwable {
        ... 

        NativeCrypto.X509_free(mContext);
        ...
    }
}

public class OpenSSLX509Certificate extends X509Certificate {

    private final long mContext;

    @Override
    protected void finalize() throws Throwable {
        ... 
        NativeCrypto.X509_free(mContext);
        ...
    }
}
public class OpenSSLX509Certificate extends X509Certificate {

    private final long mContext;  \(\text{(1)}\) SERIALIZABLE

    @Override
    protected void finalize() throws Throwable {
        ...  
        NativeCrypto.X509_free(mContext);
        ...
    }

}
public class OpenSSLX509Certificate extending X509Certificate {

    private final long mContext;

    @Override
    protected void finalize() throws Throwable {
        ...
        NativeCrypto.X509_free(mContext);
        ...
    }

}
NativeCrypto.X509_free(mContext)

X509_free(x509);  // x509 = mContext
ASN1_item_free(x509, ...)
asn1_item_combine_free(&val, ...)  // val = *pval = mContext

if (asn1_do_lock(pval, -1,...) > 0)
return;

// Decreases a reference counter (mContext+0x10),
// MUST be POSITIVE INTEGER (MSB=0)
ref = mContext + 0x10
if (*ref > 0)
    
    *ref--
else
    free(…)

Arbitrary Decrement
Proof-of-Concept Exploit

Arbitrary Code Execution in system_server
Exploit Outline

MALWARE

Malicious Serialized Object(s) w/ payload buffer

system_server
Exploit Outline

MALWARE

system_server

shellcode
First Step of the Exploit

Own the Program Counter (PC)
Creating an Arbitrary Code Exec Exploit

ARSENAL

1. Arbitrary Decrement
2. Controlled Buffer
Constrained Arbitrary Memory Overwrite
Constrained Arbitrary Memory Overwrite

Bundle

OpenSSLX509Certificate
mContext=0x11111100

OpenSSLX509Certificate
mContext=0x11111100

... 

OpenSSLX509Certificate
mContext=0x11111100

\* 0x11111110 -\(= n\)
Constrained Arbitrary Memory Overwrite

If we knew the original value: Arbitrary Overwrite

* 0x11111110 −= n
Why Constraint Overwrite?

1. We are limited to positive values.

2. Inefficiency.
One Class to Rule Them All

Or Peles, Roe Hay
IBM Security
{orpelso,roeh}@il.ibm.com

Abstract

We present previously unknown high severity vulnerabilities in Android. The first is in the Android Platform and Google Play Services. The Platform instance affects Android 4.3-L, M (Preview) or 55% of Android devices at the time of writing. This vulnerability allows for arbitrary code execution in the context of many apps and services and results in elevation of privileges. In this paper we also demonstrate a PrivX64Concept exploit against the Google Nexus 5 device, that achieves code execution inside the highly privileged system_server process, and then either replaces an existing arbitrary application on the device with our own malware app or changes the device's SELinux policy. For some other devices, we are also able to gain kernel code execution by loading an arbitrary kernel module. We had responsibly disclosed the vulnerability to Android Security Team which labeled it as CVE-2015-2556 and patched Android 4.4 / 5.x / M and Google Play Services.

For the sake of completeness we also made

1. Introduction

Android is the most popular mobile operating system with 78% of the worldwide smartphone sales to end users in Q1 2015 [1]. Android apps are executed in a sandbox environment to protect both the system and the hosted applications from malware [2]. The Android sandbox relies on the Linux kernel’s isolation facilities. While sandboxing is a central security feature, it comes at the expense of interoperability. In many common situations, apps require the ability to interact. For example, the browser app should be capable of launching the Google Play app if the user points toward the Google Play website. To recover interoperability, a key aspect of the Android architecture is Inter-App Communications (IAC), which enables modular design and reuse of functionality across apps and app components. The Android IAC model is implemented as a message-passing system, where messages are encapsulated by Java objects. Though Java, an app (or app component) can utilize functionality exposed by another
Creating an Arbitrary Code Exec Exploit

**ARSENAL**

1. Arbitrary Decrement
2. Controlled Buffer
3. Arbitrary Overwrite*

* If we knew the original value
Creating an Arbitrary Code Exec Exploit

<table>
<thead>
<tr>
<th>ARSENAL</th>
<th>DEFENSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Arbitrary Decrement</td>
<td>1. ASLR</td>
</tr>
<tr>
<td>2. Controlled Buffer</td>
<td>2. RELRO</td>
</tr>
<tr>
<td></td>
<td>4. SELinux</td>
</tr>
</tbody>
</table>

* If we knew the original value
Finding the Original Value: bye-bye ASLR

fork without execve = no ASLR!

One Class to Rule Them All (Or Peles & Roee Hay, USENIX WOOT '15)
Creating an Arbitrary Code Exec Exploit

<table>
<thead>
<tr>
<th>ARSENAL</th>
<th>DEFENSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Arbitrary Decrement</td>
<td>1. ASLR</td>
</tr>
<tr>
<td>2. Controlled Buffer</td>
<td>2. RELRO</td>
</tr>
<tr>
<td></td>
<td>4. SELinux</td>
</tr>
</tbody>
</table>

* If we knew the original value
Using the Arbitrary Overwrite

Goal
Overwrite some pointer

Problem
.got is read only (RELRO)
A Good Memory Overwrite Target

A function pointer under `.data` `id_callback` in `libcrypto`

Called during `deserialization` of: `OpenSSLECCPrivateKey`
Triggering `id_callback` remotely

Malware

Bundle

OpenSSLECPrivateKey

BAD DATA that leads to the right path

system_server
First step accomplished

We now own the Program Counter
Creating an Arbitrary Code Exec Exploit

ARSENAL

1. Arbitrary Decrement
2. Controlled Buffer
3. Arbitrary Overwrite*

DEFENSES

1. ASLR
2. RELRO
3. Non-Executable pages
4. SELinux

* If we know the original value
Next Steps of the PoC Exploit (simplified)

system_server

`pc → r-x code`
`sp → rw- stack`
`rw- ROP chain`
`rw- shellcode`
Problem 1: SP does not point at ROP chain

```
system_server

pc  →  r-x  code
sp  →  rw-  stack
rw-  ROP  chain
rw-  shellcode
```
Solution: Stack Pivoting

Our buffer happens to be pointed by fp.
The Gadget: `mov sp, fp; ..., pop {...}`

Gadget:
Stack Pivot

\[ pc \rightarrow r-x \text{ code/pivot} \]
\[ sp \rightarrow rw- \text{ stack} \]
\[ fp \rightarrow rw- \text{ ROP chain} \]
\[ rw- \text{ shellcode} \]
Solution: Stack Pivoting

Our buffer happens to be pointed by fp. The Gadget: `mov sp, fp; ..., pop {...}`
Allocating RWX Memory

Gadget: Stack Pivot

Gadget: mmap/RWX

system_server

pc → r-x code/mmap
rw- stack
sp → rw- ROP chain
rw- shellcode
Problem 2: SELinux should prohibit mmap/RWX

- pc → r-x code/mmap
- rw- stack
- sp → rw- ROP chain
- rw- shellcode

Gadget: Stack Pivot
Gadget: mmap/RWX
Solution: Weak SELinux Policy for `system_server`

- Gadget: Stack Pivot
- Gadget: `mmap/RWX`
- `pc \rightarrow r-x code/mmap`
- `rw- stack`
- `sp \rightarrow rw- ROP chain`
- `rw- shellcode`
Solution: Weak SELinux Policy for system_server

allow system_server self:process execmem
Allocating RWX Memory

Gadget: Stack Pivot

Gadget: mmap/RWX

system_server

pc → r-x code/mmap
rw- stack
sp → rw- ROP chain
rw- shellcode
rwx -
Copying our Shellcode

Gadget: Stack Pivot

Gadget: memcpy

Gadget: mmap/RWX

system_server

pc → r-x code/memcpy
rw- stack
sp → rw- ROP chain
rw- shellcode
rwx -
Copying our Shellcode

Gadget: Stack Pivot

Gadget: memcpy

Gadget: mmap/RWX

system_server

pc → r-x code/memory
rw- stack
sp → rw- ROP chain
rw- shellcode
rwx shellcode
Executing our Shellcode

<table>
<thead>
<tr>
<th>Gadget: Stack Pivot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadget: memcpy</td>
</tr>
<tr>
<td>Gadget: mmap/RWX</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

shellcode

system_server

code
stack
ROP chain
shellcode

sp → rw- stack
rw- shellcode
pc → rwx shellcode
Creating an Arbitrary Code Exec Exploit

<table>
<thead>
<tr>
<th>ARSENAL</th>
<th>DEFENSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Arbitrary Decrement</td>
<td>1. ASLR</td>
</tr>
<tr>
<td>2. Controlled Buffer</td>
<td>2. RELRO</td>
</tr>
<tr>
<td></td>
<td>4. SELinux</td>
</tr>
</tbody>
</table>

* If we know the original value
Shellcode

Runs as system, still subject to the SELinux, but can:

- Replacement of apps
- SELinux bypass
- Access to apps' data
- Kernel code execution*

* On select devices
One Class to Rule Them All

Exploiting CVE-2015-3825 for Replacing an Existing App on the Device

USENIX WOOT '15

Or Peles & Roee Hay
IBM Security
Google’s Patch for CVE-2015-3825

public class OpenSSLX509Certificate extends X509Certificate {

    private final long mContext;

    ...

}
Google’s Patch for CVE-2015-3825

public class OpenSSLX509Certificate extends X509Certificate {
  
  private transient final long mContext;
  
  ...

}
Our 2\textsuperscript{nd} Contribution: Vulnerabilities in SDKs

Finding Similar Vulnerabilities in SDKs

**Goal.** Find vulnerable *Serializable* classes in 3rd-party SDKs.

**Why.** Fixing the Android Platform Vulnerability is not enough.
Analyzed over 32K of popular Android apps using dexlib2.

|----------------|-----------|------------|
Root Cause (for most of the SDKs)

SWIG, a C/C++ to Java interoperability tool, can generate vulnerable classes.

```java
public class Foo implements Bar {
    private long swigCPtr;
    protected boolean swigCMemOwn;
    ...
    protected void finalize() {
        delete();
    }
    public synchronized void delete() {
        ...
        exampleJNI.delete_Foo(swigCPtr);
        ...
    }
    ...
}
```
Wrap-up
Summary

- Found a high severity vulnerability in Android (Exp. 1).
- Wrote a reliable PoC exploit against it
- Found similar vulnerabilities in 6 third-party SDKs (Exp. 2)
- Patches are available for all of the vulnerabilities and also for SWIG.
Statement of Good Security Practices: IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a lawful, comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM DOES NOT WARRANT THAT ANY SYSTEMS; PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY.

THANK YOU

www.ibm.com/security