One Class to Rule Them All

APPSEC IL '15

Or Peles & Roee Hay
IBM Security X-Force
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 & Roe Hay, OWASP APPSEC IL '15)
Introduction
Threat Model
Intent { play:// ...}
Malware Attack via Inter-App Communication

Intent { play:// …}
An Intent can also contain

**Bundle**

**SIMPLE OBJECTS**
An Intent can also contain

Bundle

- SIMPLE OBJECTS
- Serializable OBJECTS
Motivation
One Class to Rule Them All (Or Peles & Roee Hay, OWASP APPSEC IL ’15)

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

One Class to Rule Them All (Or Peles & Roeo Hay, OWASP APPSEC IL '15)
The Serialized Object

```java
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 \rightarrow 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) {
    unparsel(); \(\leftrightarrow\) 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.

MALWARE  ----->  SYSTEM_SERVER

Executed Object
(4) Make it Execute some Sensitive Code

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

```java
class Foo implements Serializable {
    private long mObject;
    ...
    private native final destroy();
    @Override
    protected void finalize() throws Throwable {
        try { destroy(); }
        finally { super.finalize(); }
    }
}
```
Experiment 1

boot.art
Experiment 1

boot.art

~13K Loadable Java Classes
One Class to Rule Them All (Or Peles & Roe Hay, OWASP APPSEC IL '15)

Experiment 1

boot.art

~13K Loadable Java Classes

App: Loaded classes using Reflection
Experiment 1

boot.art

~13K Loadable Java Classes

App: Loaded classes using Reflection

Dumped classes:
1. Serializable
2. Finalize method
3. Controllable fields

One Class to Rule Them All (Or Peles & Roeee Hay, OWASP APPSEC IL '15)
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);
    ...
}

/** (1) SERIALIZABLE */
public class OpenSSLX509Certificate
extends X509Certificate {

    private final long mContext;  

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

    }

}
public class OpenSSLX509Certificate {

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

    @Override
    protected void finalize() throws Throwable { \(\text{CONTROLLABLE PTR} \tag{2}\)

        NativeCrypto.X509_free(mContext); \(\text{EXECUTED AUTOMATICALLY BY THE GC} \tag{3}\)

    }

}
NativeCrypto.X509_free(mContext)

X509_free(x509);  // x509 = mContext

ASN1_item_free(x509, ...)

asn1_item_combine_free(&val, ...)

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

Malicious Serialized Object(s) w/ payload buffer

MALWARE

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

Bundle

OpenSSLX509Certificate
mContext=0x11111100

OpenSSLX509Certificate
mContext=0x11111100

...
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

Bundle

OpenSSLX509Certificate
mContext=0x111111100

OpenSSLX509Certificate
mContext=0x111111100

...
Why Constraint Overwrite?

1. We are limited to positive values.
2. Inefficiency.
One Class to Rule Them All  (Or Peles & Roee Hay, OWASP APPSEC IL ’15)

Solution in the Paper

ONE CLASS TO RULE THEM ALL
0-DAY DESERIALIZATION VULNERABILITIES IN ANDROID

Or Peles, Roee Hay
IBM Security
{orpalea, roeeh}@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.x.5, M (Preview 1) 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 ProxiedConcept exploit against the Google Nexus 5 device, that achieves code execution inside the high 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-2015-2020 (internally as ANDROID-25437603/25838090) and patched Android 4.4.5/5.x.1 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 sandboxed 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 in a message passing system, where messages are encapsulated by Java objects. Through IExchanges, 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

One Class to Rule Them All (Or Peles & Roei Hay, OWASP APPSEC IL '15)
Creating an Arbitrary Code Exec Exploit

**ARSENAL**

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

* If we knew the original value

**DEFENSES**

1. ASLR
2. RELRO
3. Non-Executable pages
4. SELinux
Finding the Original Value: bye-bye ASLR

fork without execve = no ASLR!
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 knew the original value
Using the Arbitrary Overwrite

Goal
Overwrite some pointer

Problem
.got is read only (RELRO)
A function pointer under `.data` id_callback in `libcrypto` called during deserialization of: `OpenSSLECPrivateKey`
Triggering `id_callback` remotely

Malware

Bundle

OpenSSLECPublicKey

BAD DATA that leads to the right path

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

## ARSENAL
1. Arbitrary Decrement
2. Controlled Buffer
3. Arbitrary Overwrite*  
   * If we know the original value

## DEFENSES
1. ASLR
2. RELRO
3. Non-Executable pages
4. SELinux
Next Steps of the PoC Exploit (simplified)

One Class to Rule Them All (Or Peles & Roeie Hay, OWASP APPSEC IL '15)
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 {...}

The Gadget:

Stack Pivot

Gadget: Stack Pivot

system_server

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

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

system_server

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

Gadget: Stack Pivot

Gadget: mmap/RWX

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

One Class to Rule Them All (Or Peles & Roeen Hay, OWASP APPSEC IL '15)
Solution: Weak SELinux Policy for system_server

Gadget: Stack Pivot

Gadget: mmap/RWX

system_server

pc → r-x code/mmap
rw- stack
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

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

Gadget: Stack Pivot

Gadget: memcopy

Gadget: mmap/RWX

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

Gadget: Stack Pivot
Gadget: memcpy
Gadget: mmap/RWX

shellcode

system_server
r-x code
rw- stack
sp → rw- ROP chain
rw- shellcode
pc → rwx shellcode
Creating an Arbitrary Code Exec Exploit

ARSENAL

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

* If we know the original value

DEFENSES

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

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

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

* On select devices

One Class to Rule Them All (Or Peles & Roe Hay, OWASP APPSEC IL ’15)
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;
    ...
}

MISSING MODIFIER
BEFORE OUR DISCLOSURE!
(NOW PATCHED)
Google’s Patch for CVE-2015-3825

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

MISSING MODIFIER BEFORE OUR DISCLOSURE!
(NOW PATCHED)

One Class to Rule Them All (Or Peles & Roee Hay, OWASP APPSEC IL '15)
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.
Experiment 2

Analyzed over 32K of popular Android apps using dexlib2.

Main Results

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