RSAConference2016
San Francisco | February 29 – March 4 | Moscone Center

Connect to Protect

SESSION ID: MBS-F03

Android Serialization Vulnerabilities Revisited

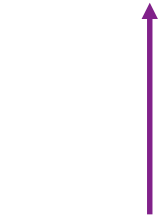
Roee Hay
X-Force Application Security Team Lead
IBM Security

@roeehay

Joint work with Or Peles

We will see how this Android SDK class

```
public class OpenSSLX509Certificate
  extends X509Certificate {

  private ____ final long mContext;
  …

}
```



MISSING MODIFIER
BEFORE OUR
DISCLOSURE!
(NOW PATCHED)

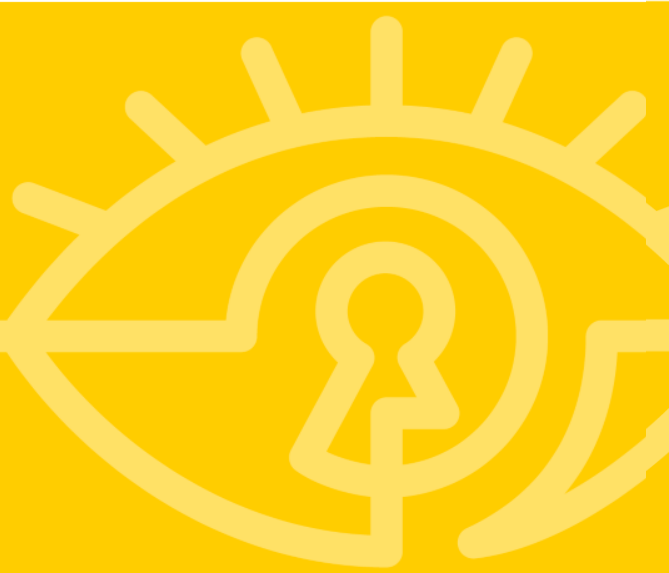
Led to malware capable of this…

REPLACEMENT OF APPS

SELINUX BYPASS

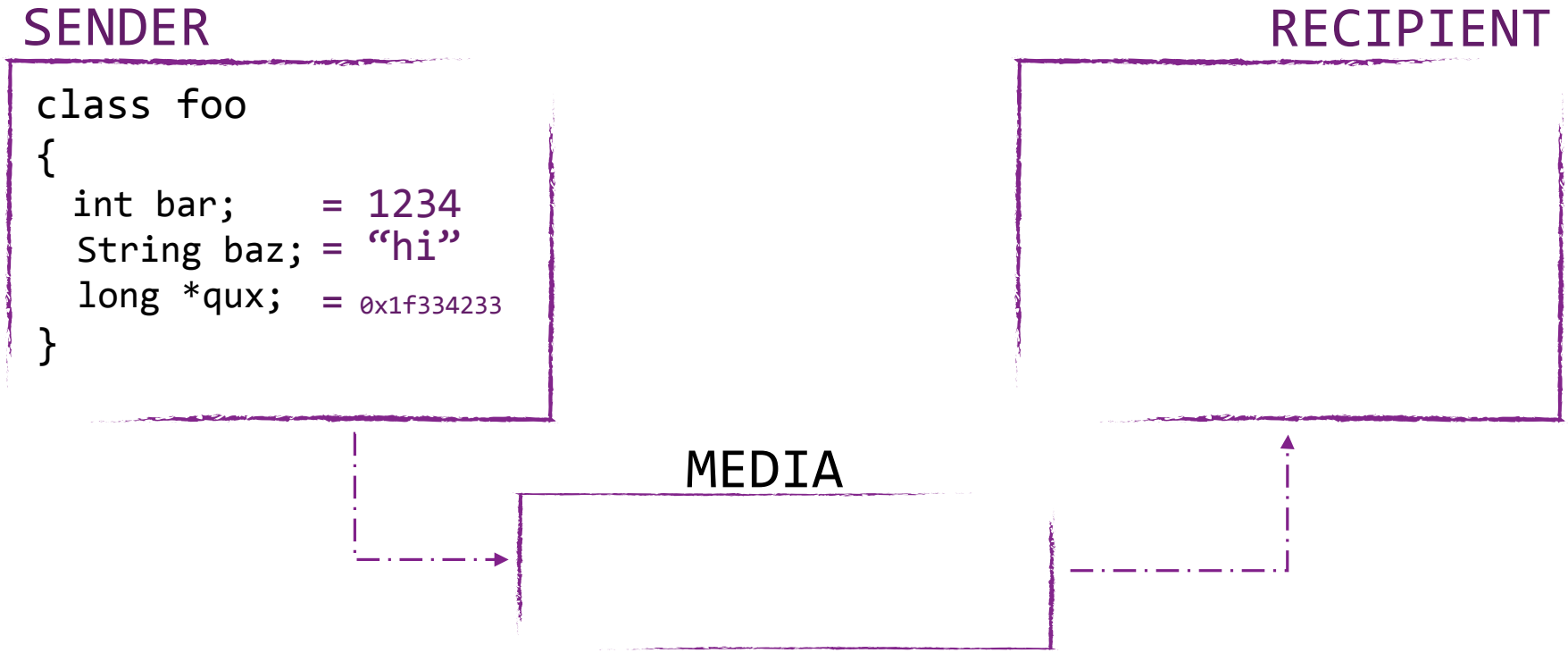
ACCESS TO APPS' DATA

KERNEL CODE EXEC
(on select devices)

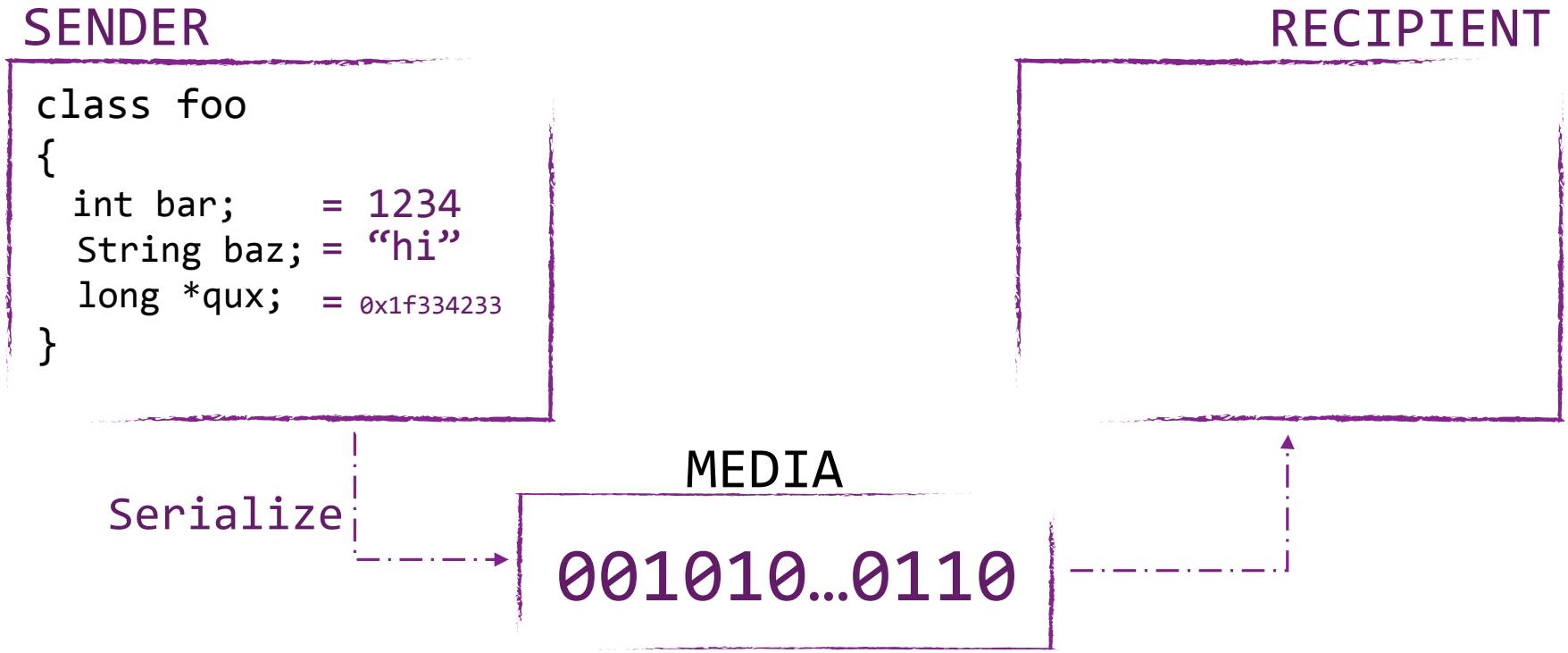


Introduction

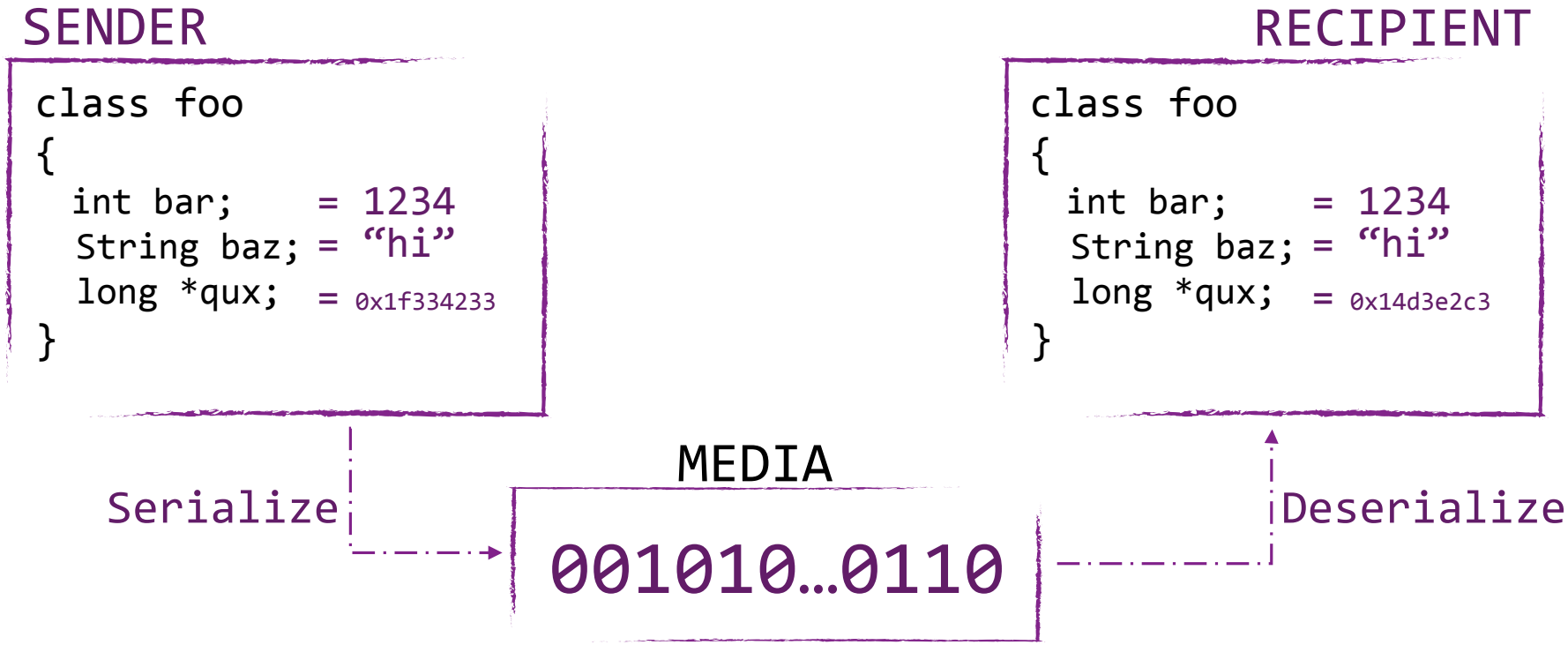
Serialization



Serialization



Serialization

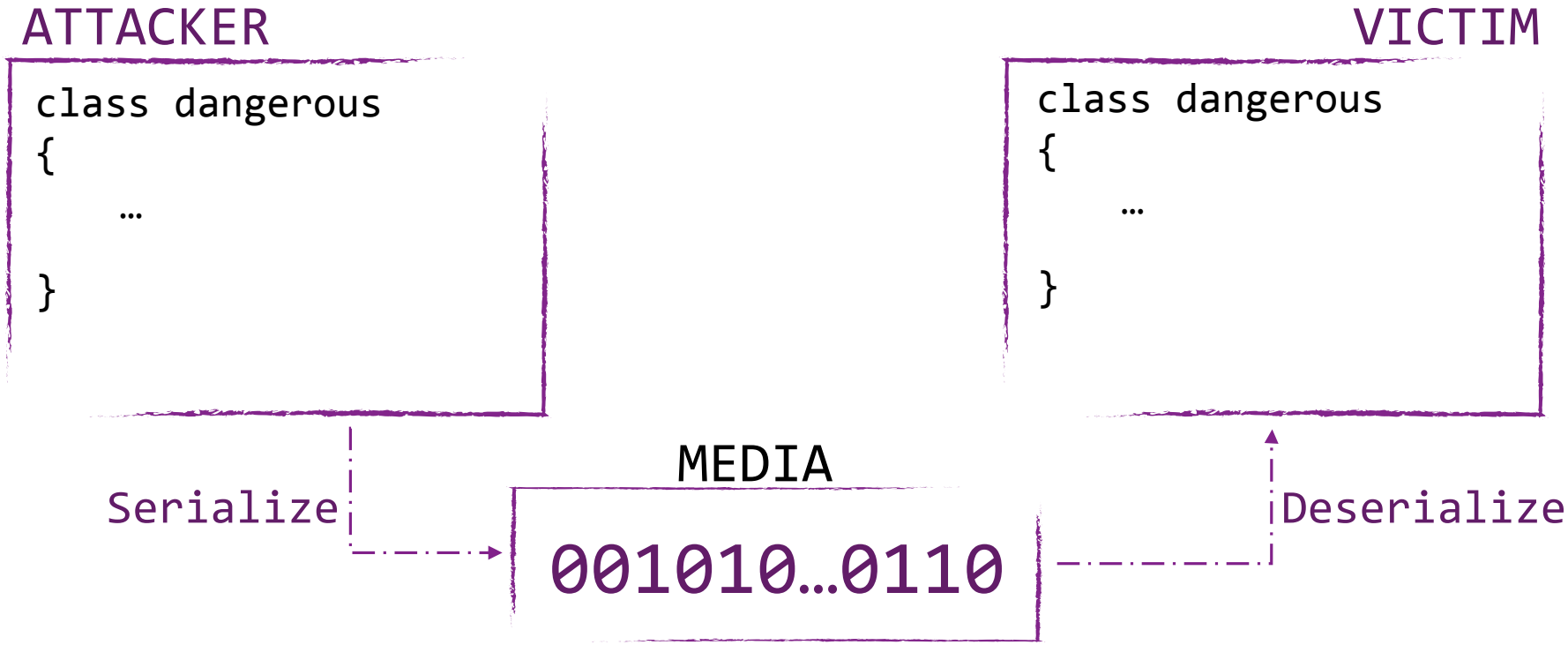


Vulnerability Root Cause

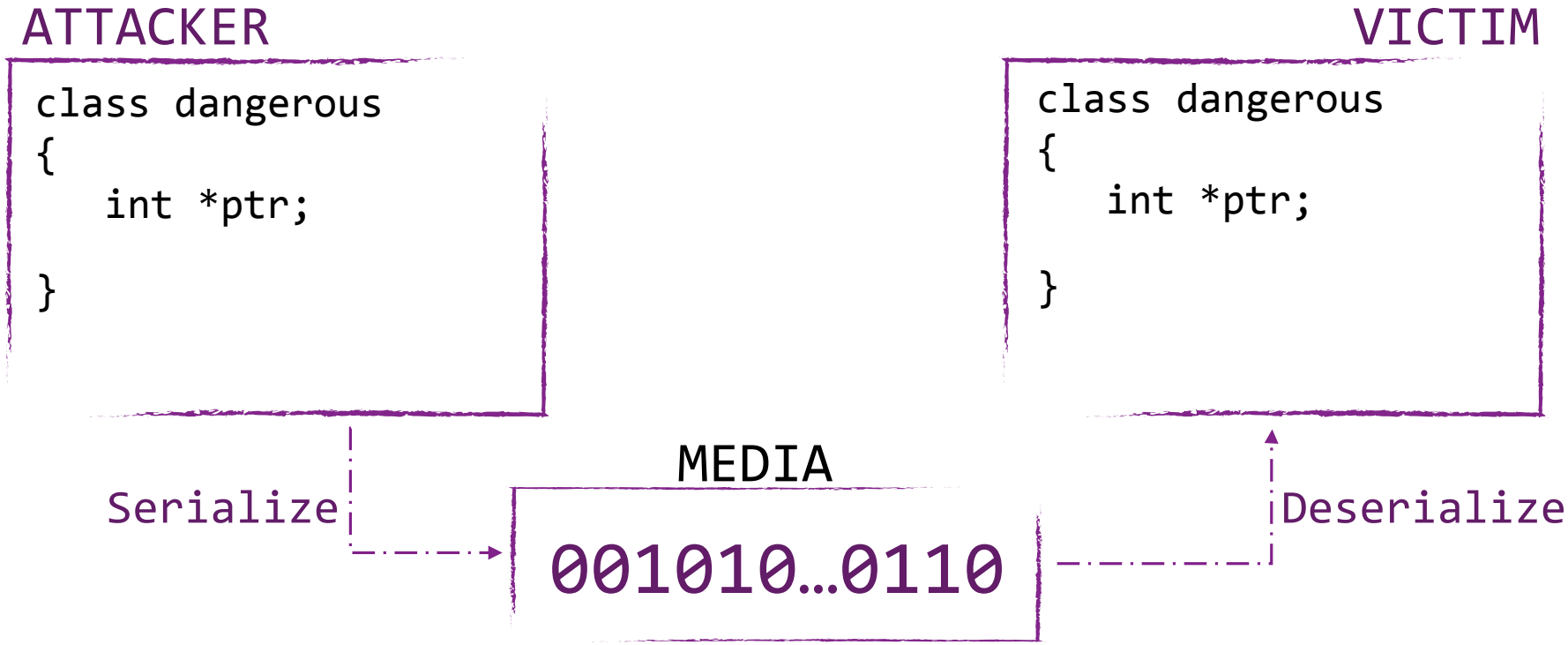
DESERIALIZATION OF UNTRUSTED DATA

```
ObjectInputStream ois =
 new ObjectInputStream(insecureSource);

Foo t = (Foo)ois.readObject();
```

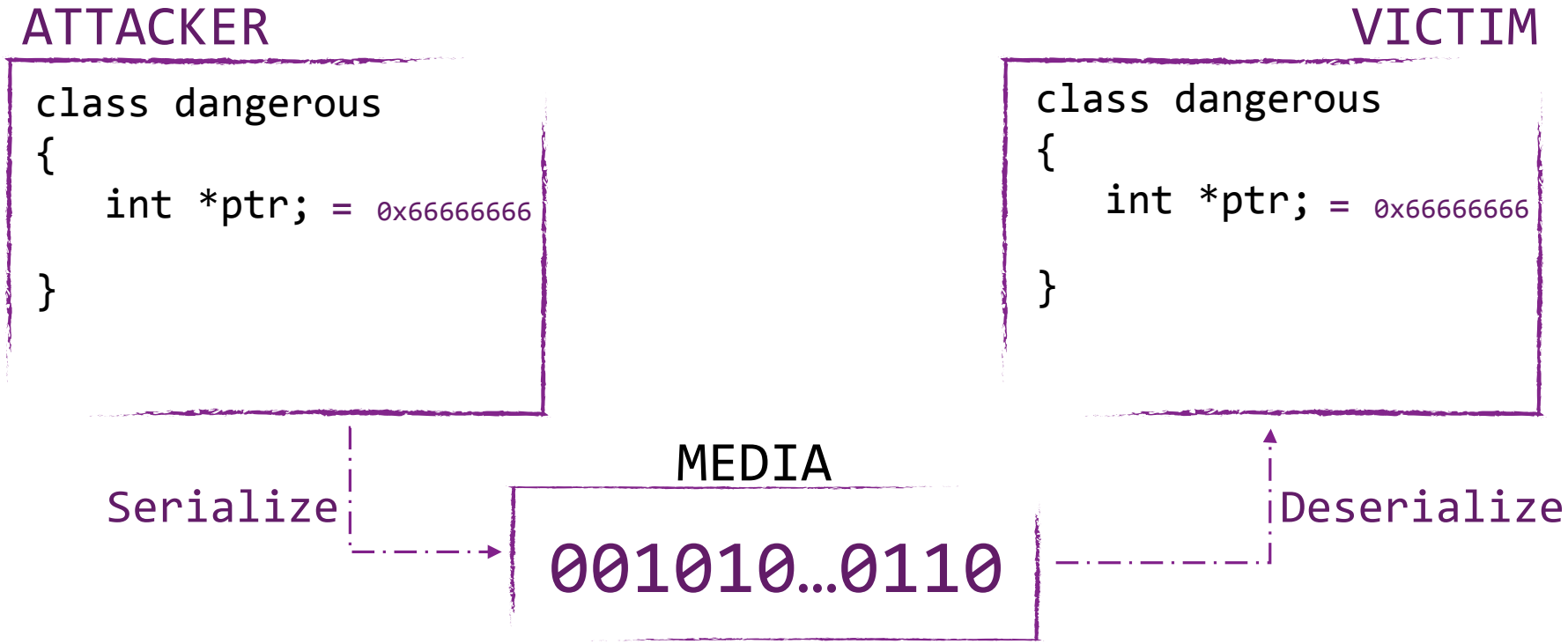
Vulnerability Root Cause



Example of a vulnerability



Example of a vulnerability



Vulnerability Root Cause

RECIPIENT CODE

```
ObjectInputStream ois =
 new ObjectInputStream(insecureSource);

dangerous t = (dangerous)ois.readObject();

callNative(t.ptr)
```

History of Serialization Vulnerabilities

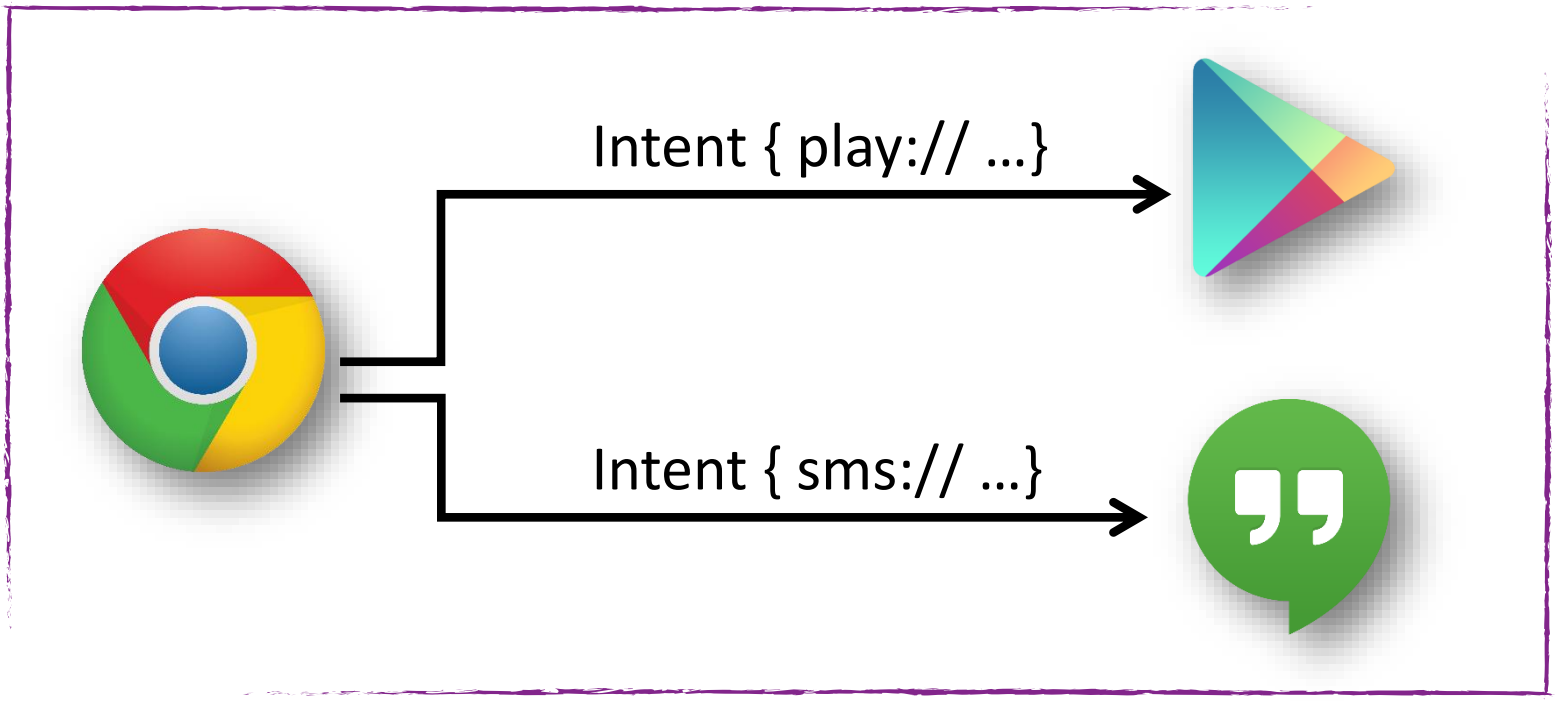
2009	- Shocking News in PHP Exploitation
2011	- Spring Framework Serialization-based remoting vulnerabilities
2012	- AtomicReferenceArray type confusion vulnerability
2013	- Apache Commons FileUpload Deserialization Vulnerability
	- Ruby on Rails YAML Deserialization Code Execution
2014	- Android <5.0 Privilege Escalation using ObjectInputStream
2015	- Android OpenSSLX509Certificate Deserialization Vulnerability
	- Apache Groovy Deserialization of Untrusted Data
	- Apache Commons Collections Unsafe Classes

History of Serialization Vulnerabilities

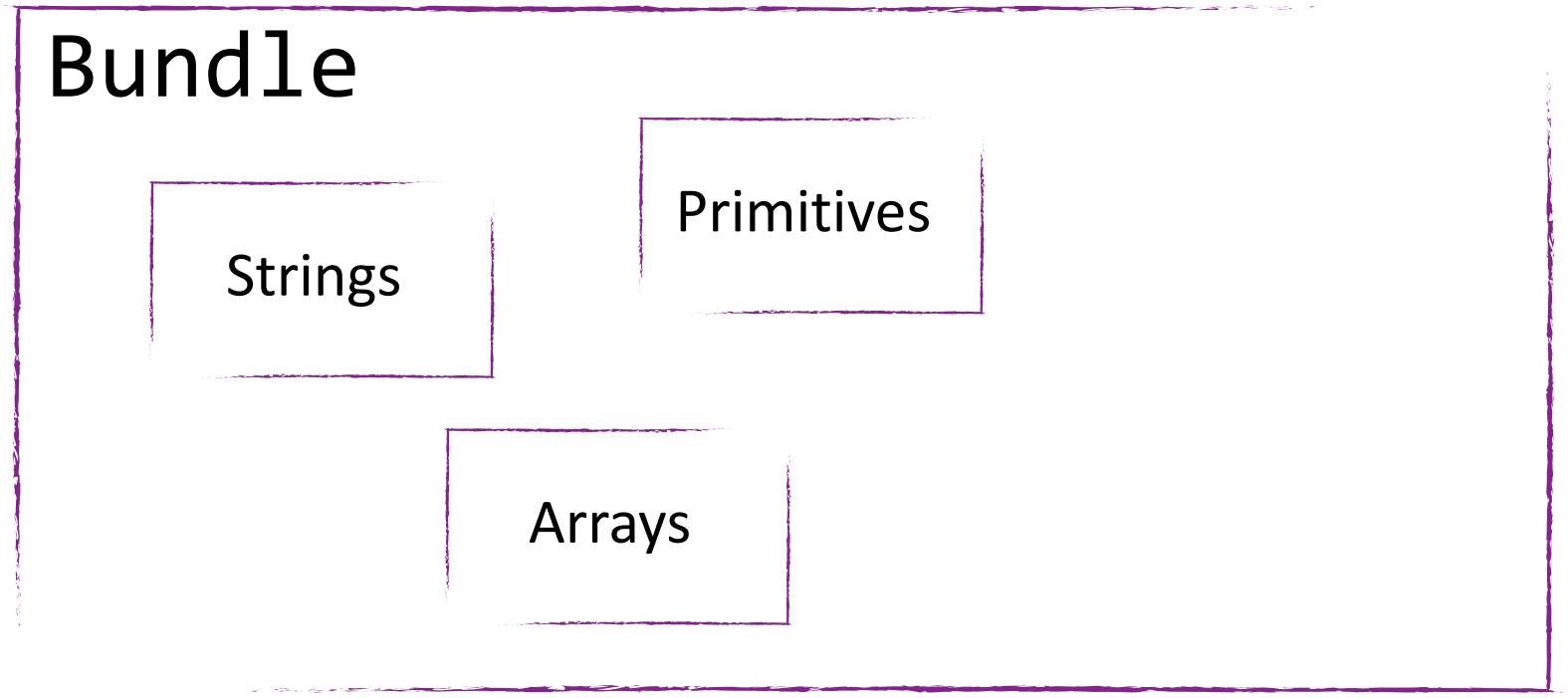
2009	- Shocking News in PHP Exploitation
2011	- Spring Framework Serialization-based remoting vulnerabilities
2012	- AtomicReferenceArray type confusion vulnerability
2013	- Apache Commons FileUpload Deserialization Vulnerability
	- Ruby on Rails YAML Deserialization Code Execution
2014	**- Android <5.0 Privilege Escalation using ObjectInputStream**
2015	**- Android OpenSSLX509Certificate Deserialization Vulnerability**
	- Apache Groovy Deserialization of Untrusted Data
	- Apache Commons Collections Unsafe Classes

Android Inter-App Communication

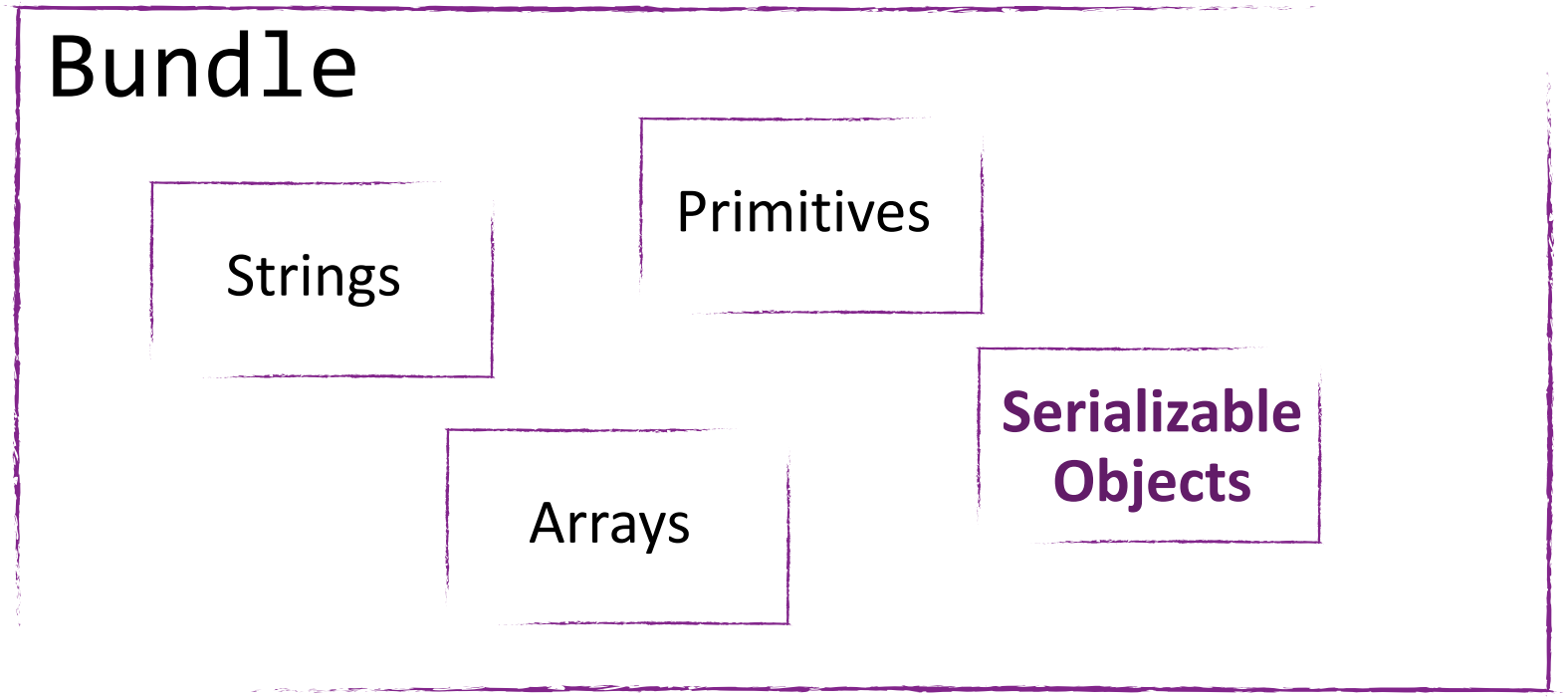
Android Inter-App Communication 101

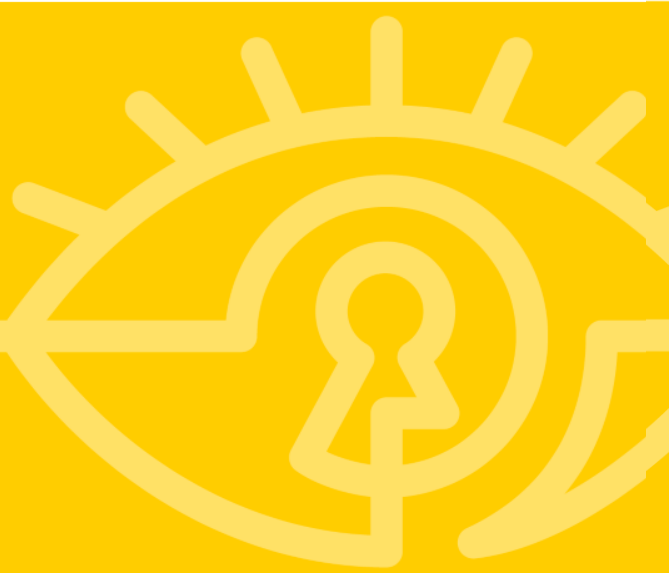


An Intent can also contain



An Intent can also contain



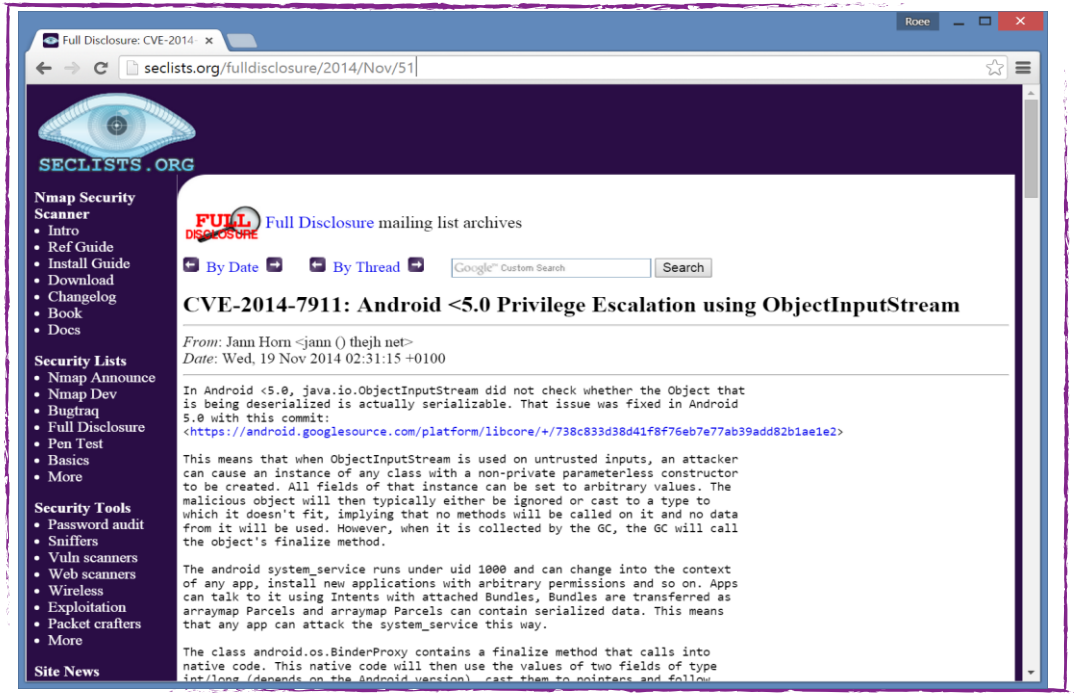


Motivation

Previous work

CVE-2014-7911 (Jann Horn):

Non-Serializable Classes can be Deserialized on target.



Exploiting CVE-2014-7911

Step 1. Find an interesting target.

MALWARE

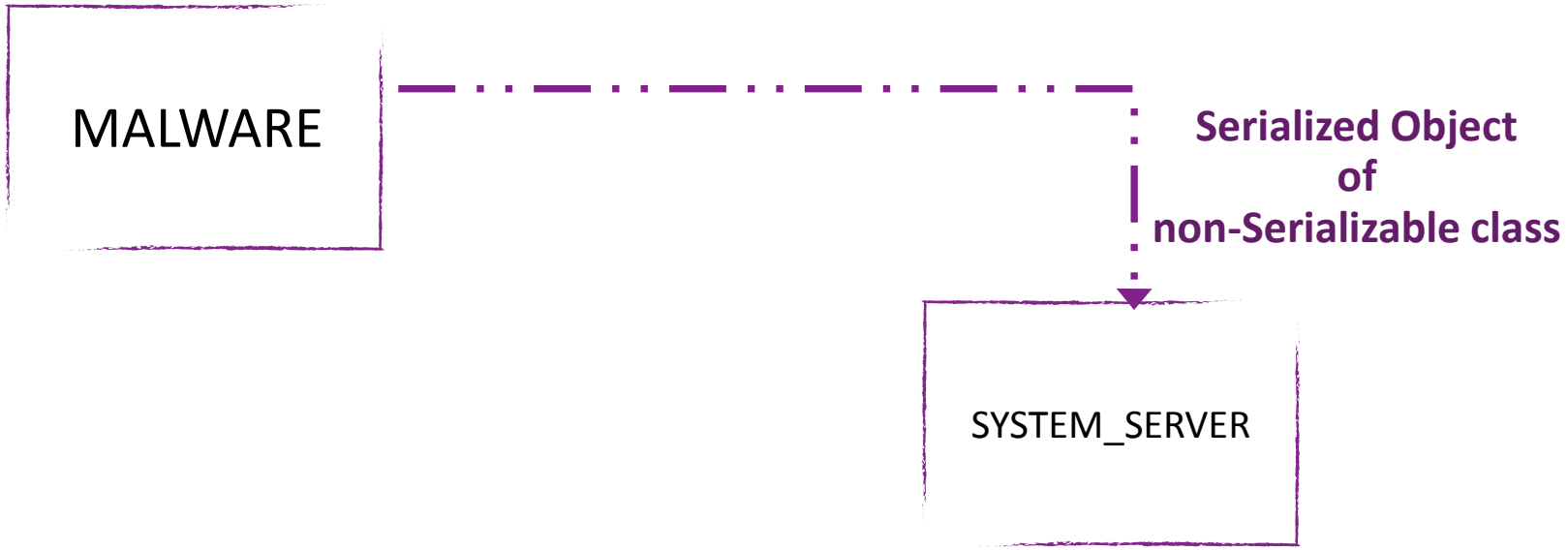
TARGET

The Target

SYSTEM_SERVER

Exploiting CVE-2014-7911

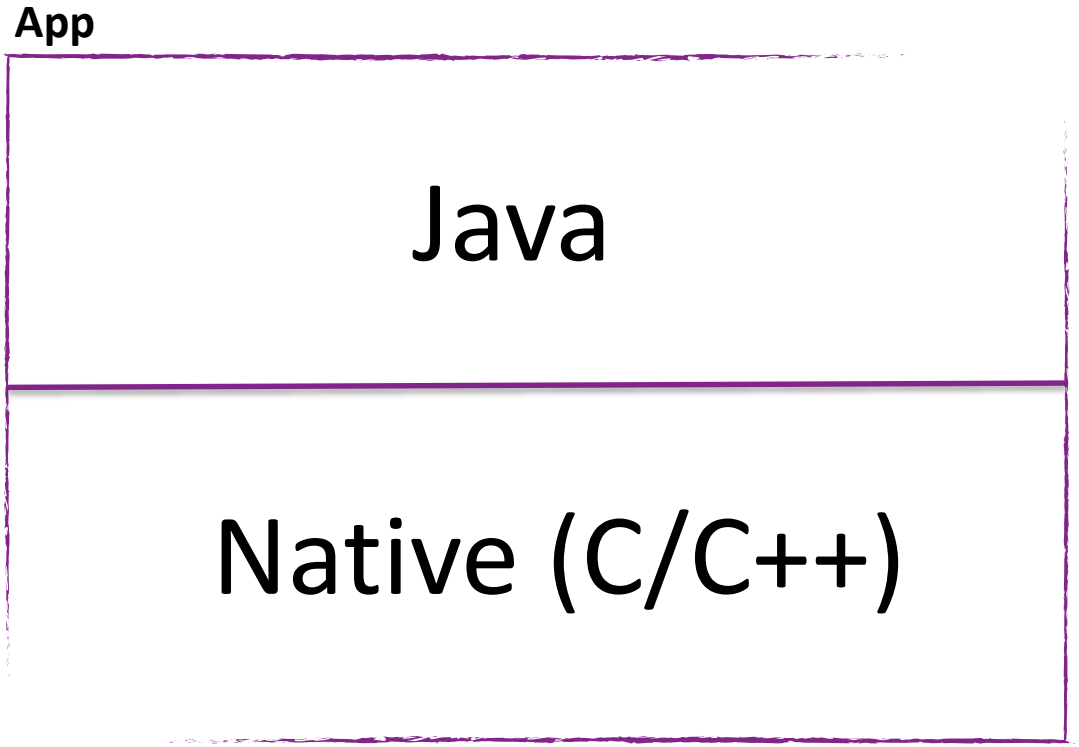
Step 2. Send target a ‘serialized’ object in a Bundle



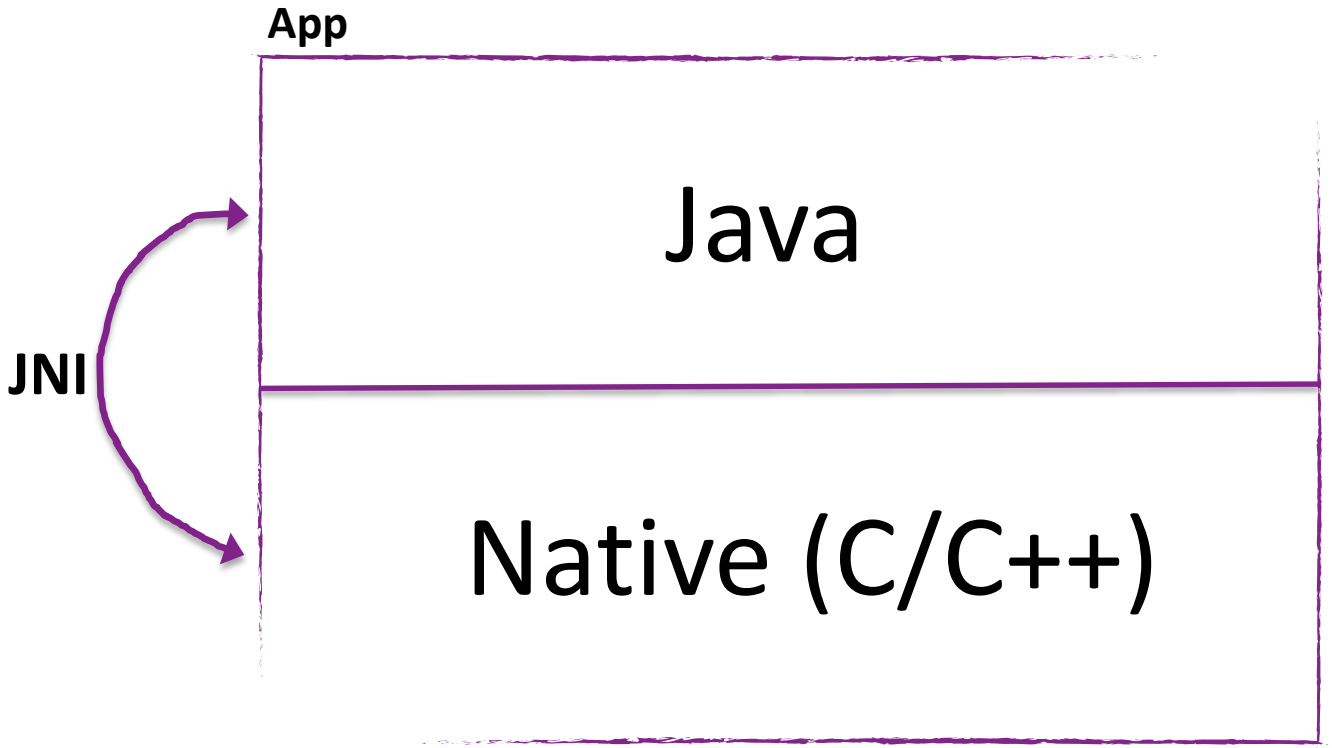
The Serialized Object


```
final class BinderProxy implements IBinder {

        private long mOrgue; ← POINTER
        …
        private native final destroy();

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

        }
```

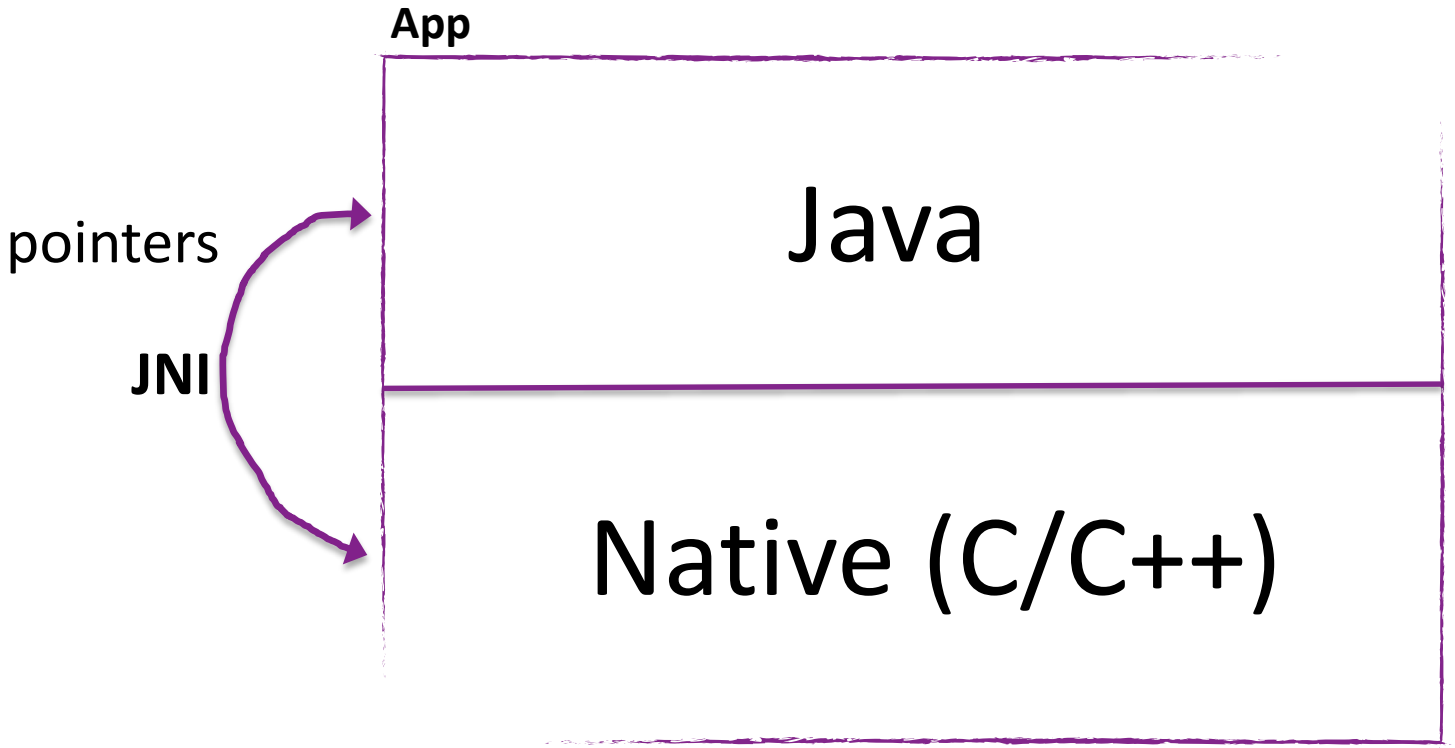
Android Apps are not just Java…



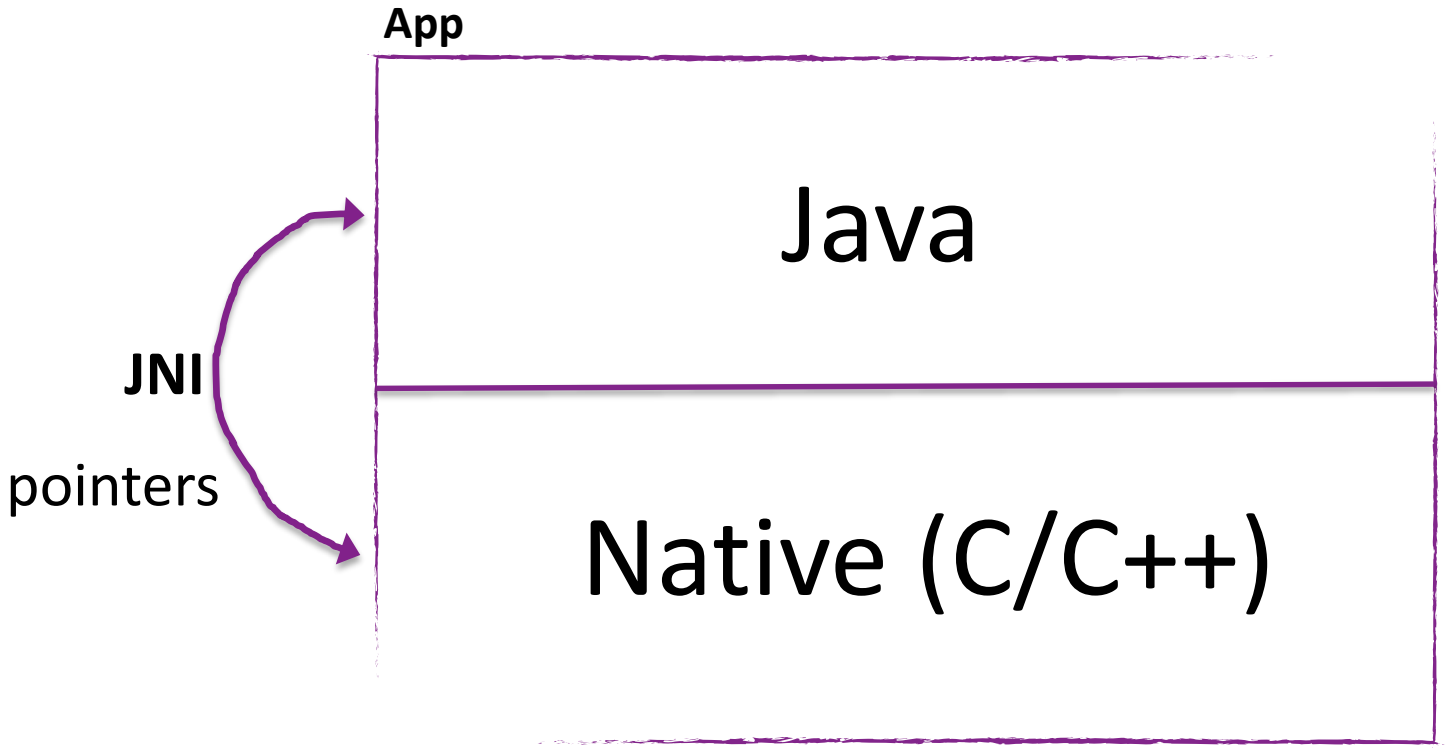
Android Apps are not just Java…



Pointers may pass back and forth



Pointers may pass back and forth

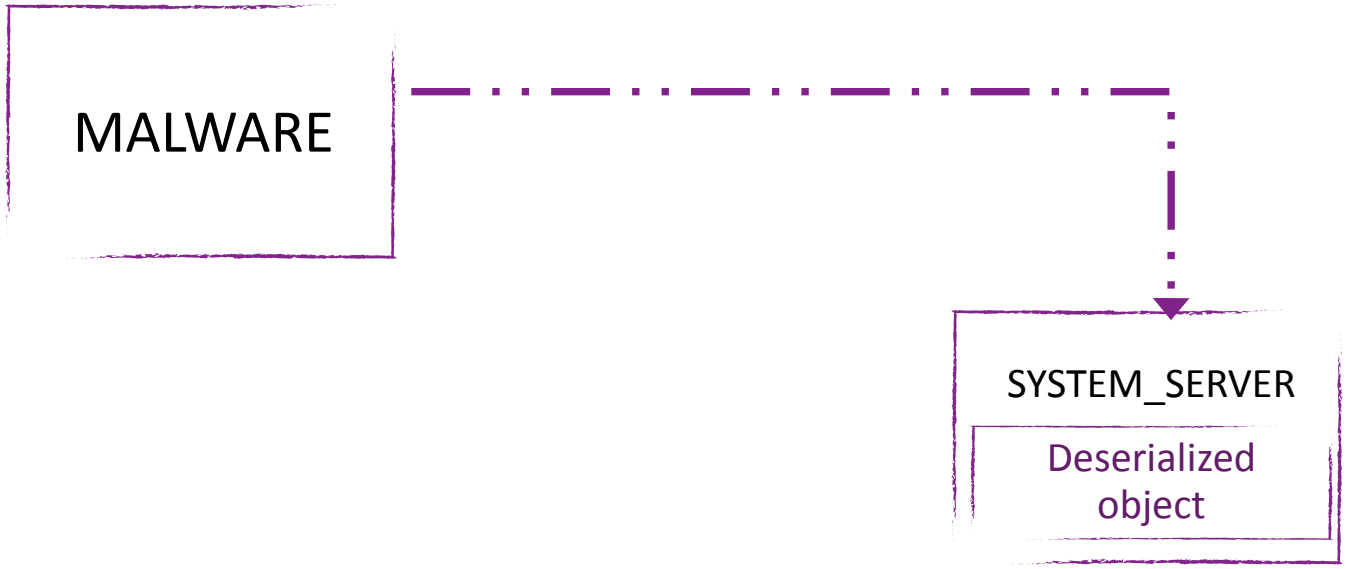


The Serialized Object


```
final class BinderProxy implements IBinder {

        private long mOrgue; ← POINTER
        …
        private native final destroy();

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

        }
```

Exploiting CVE-2014-7911

Step 3. Make it deserialize on the target



Make it deserialize automatically

All Bundle members are deserialized with a single ‘touch’ without **type checking** before deserialization

e.g.

```
public String getString(String key)
{
    unparcel(); ← DESERIALIZES ALL
    final Object o = mMap.get(key);
    try { return (String) o; }
    catch (ClassCastException e)
            {typeWarning…}
}
```

Exploiting CVE-2014-7911

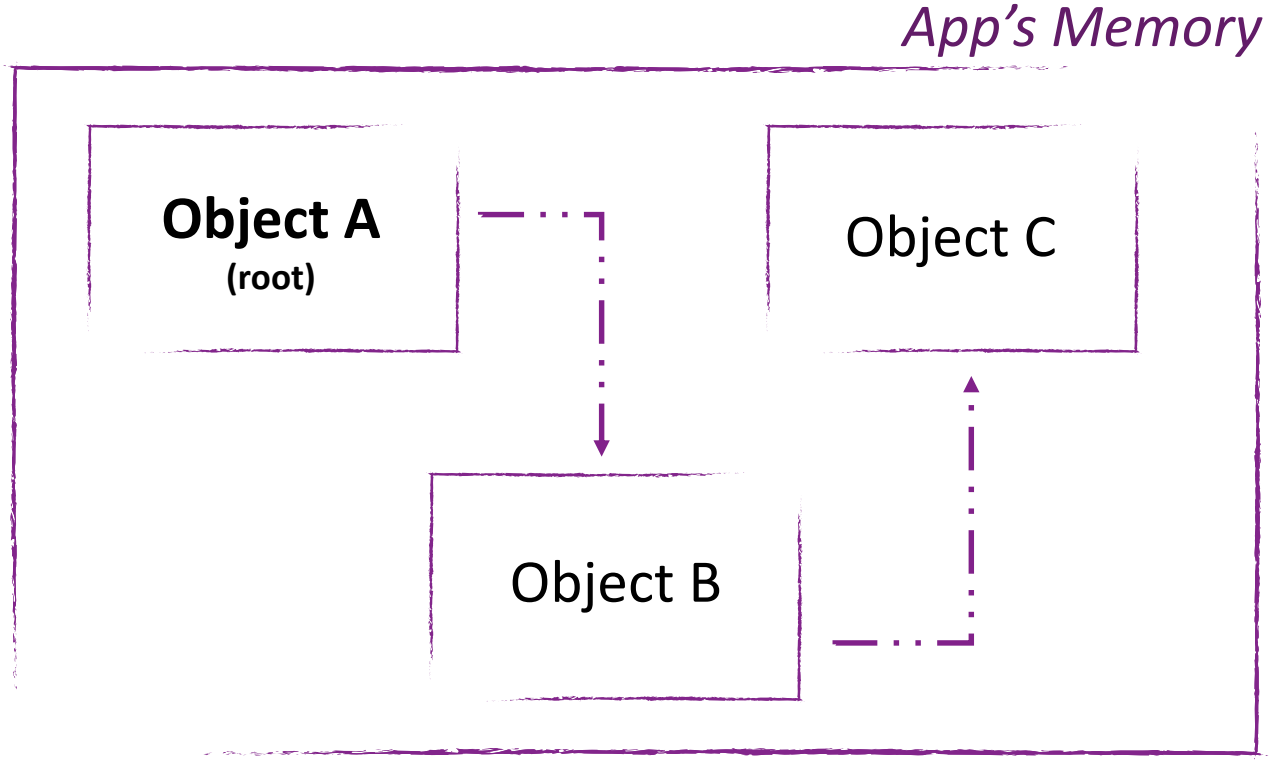
Step 4. Make one of its methods *execute* on target.

The Serialized Object

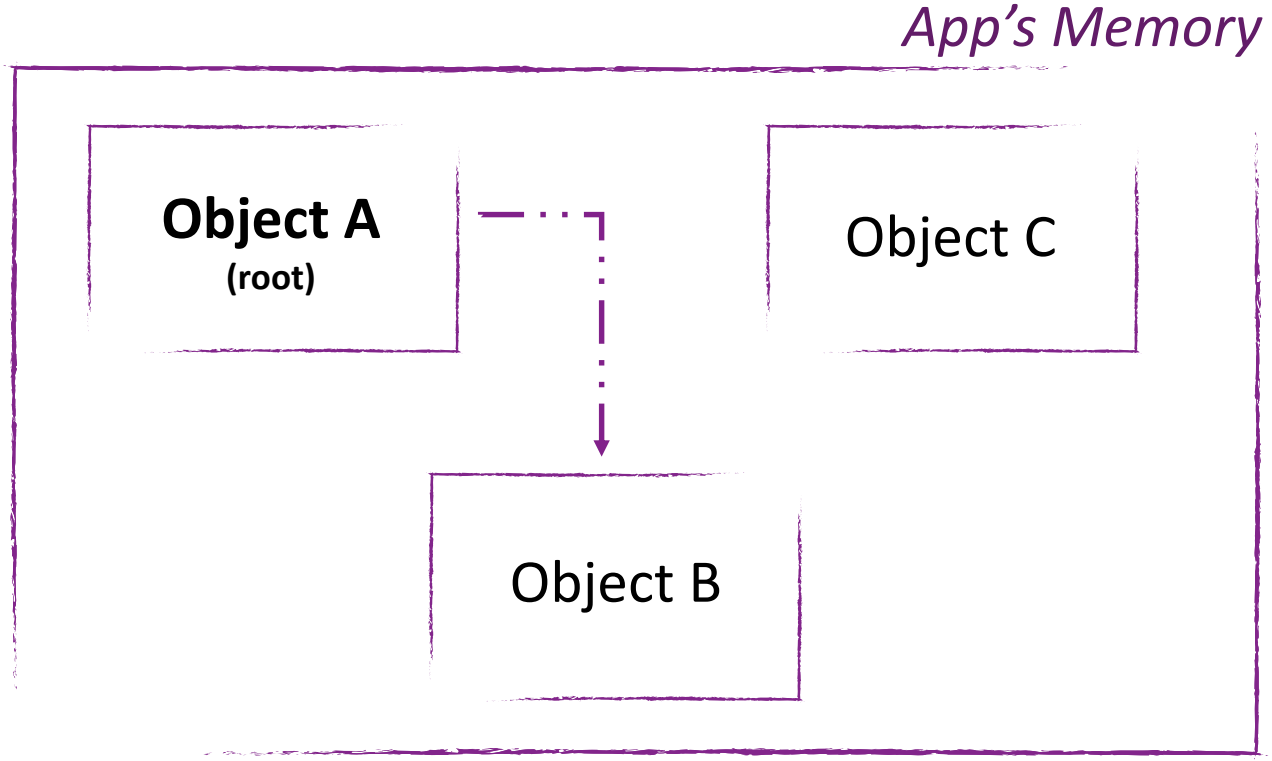

```
final class BinderProxy implements IBinder {

        private long mOrgue;
        …
        private native final destroy();

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

        }
```

← EXECUTED AUTOMATICALLY BY THE GC

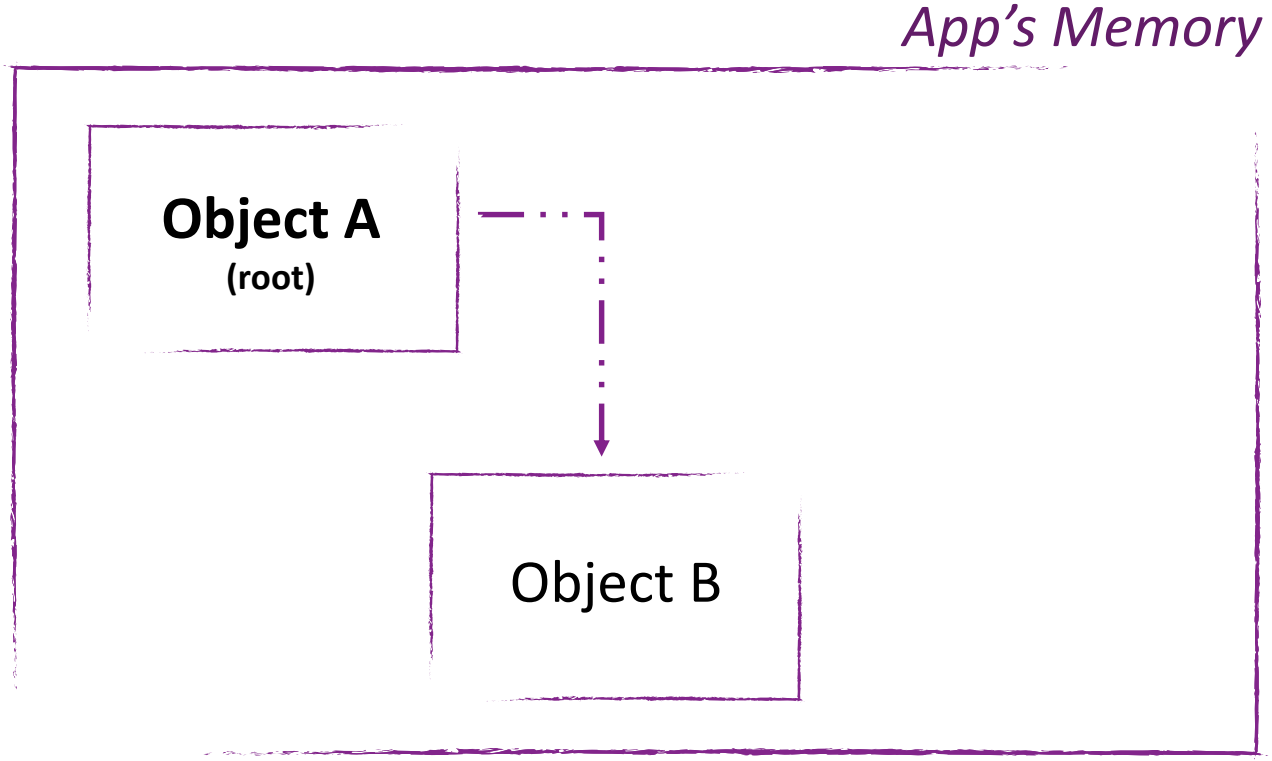
A Word about Garbage Collection



A Word about Garbage Collection



A Word about Garbage Collection



The Serialized Object

```
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

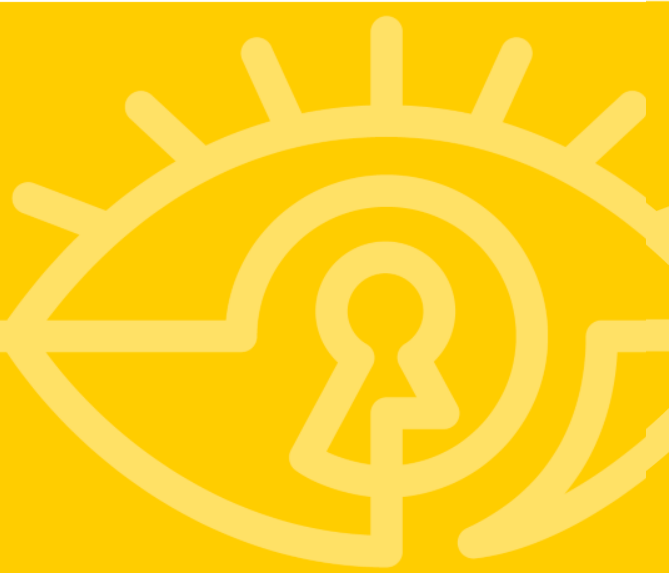
The Serialized Object

```
final class BinderProxy implements IBinder {

        private long mOrgue;
        …
        private native final destroy();

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

        }
```

← NATIVE METHOD THAT USES THE PTR

Google's Patch for CVE-2014-7911

Do not Deserialize Non-Serializable Classes



Our 1st contribution:
The Android Vulnerability

CVE-2015-3825/37

Our Research Question

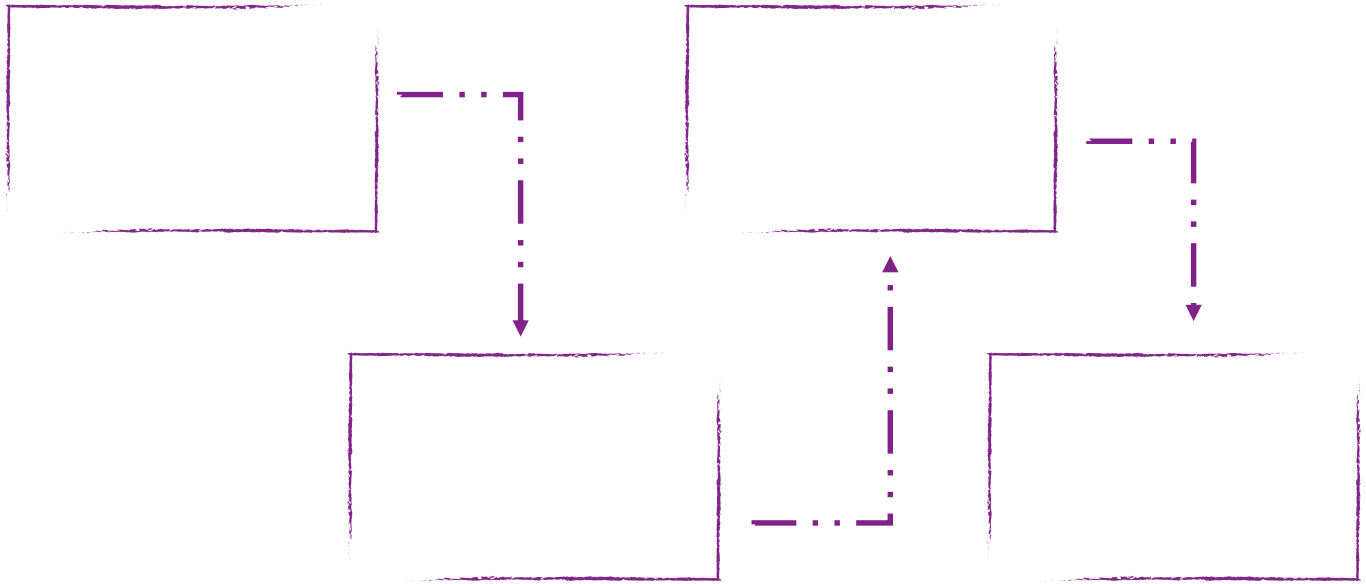

```
Class Foo implements Serializable {

  private long mObject;
  …
  private native final destroy();

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

CONTROLLABLE POINTER

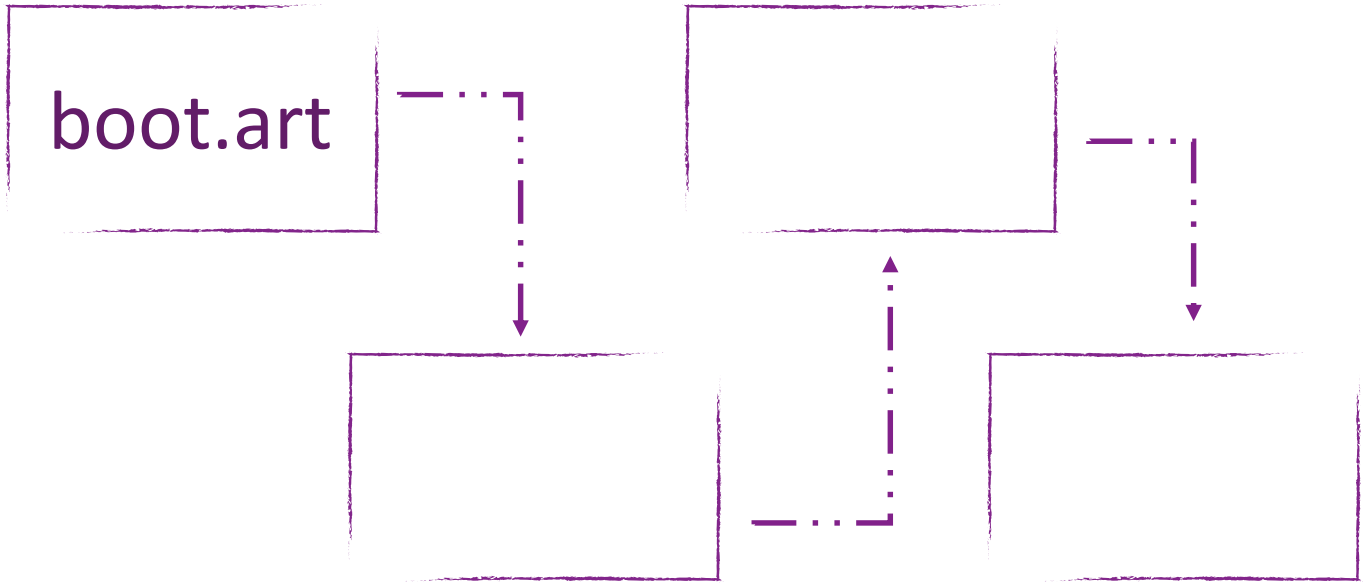
POINTER USED IN NATIVE CODE.

EXECUTED AUTOMATICALLY BY THE GC

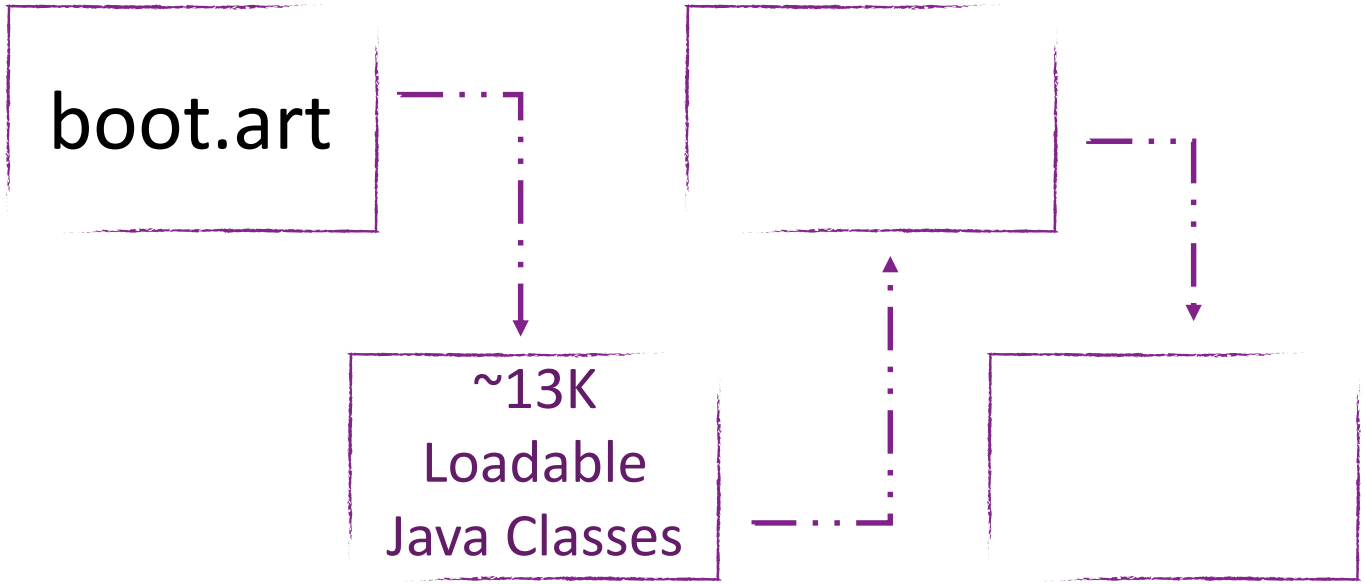
Experiment 1



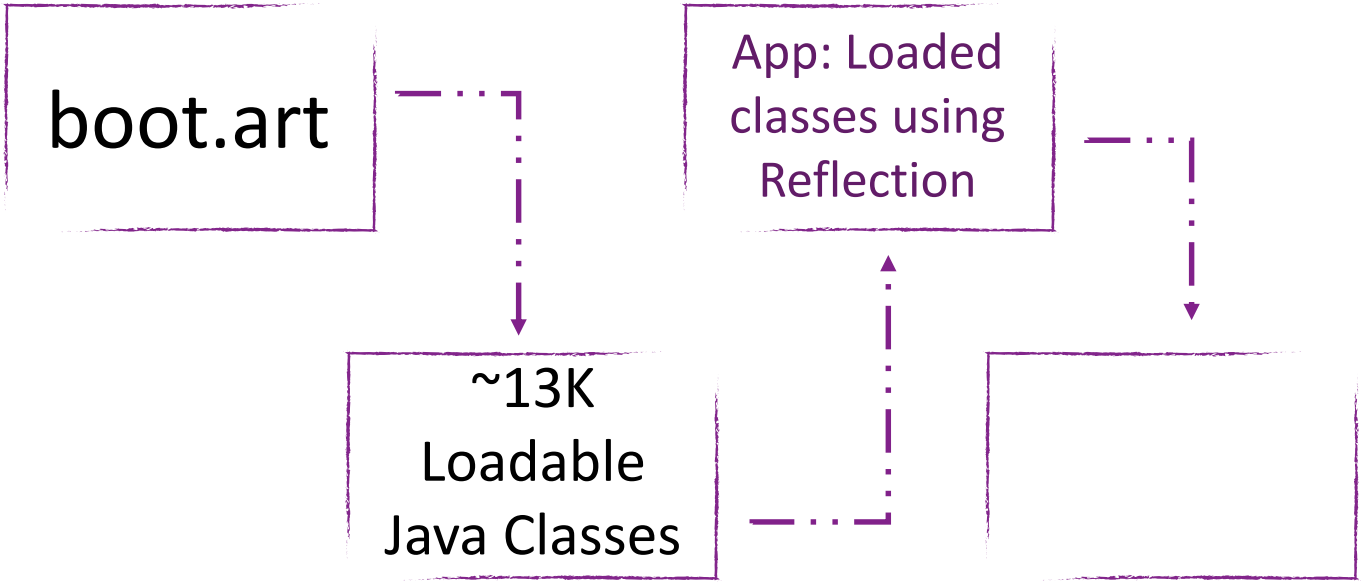
Experiment 1



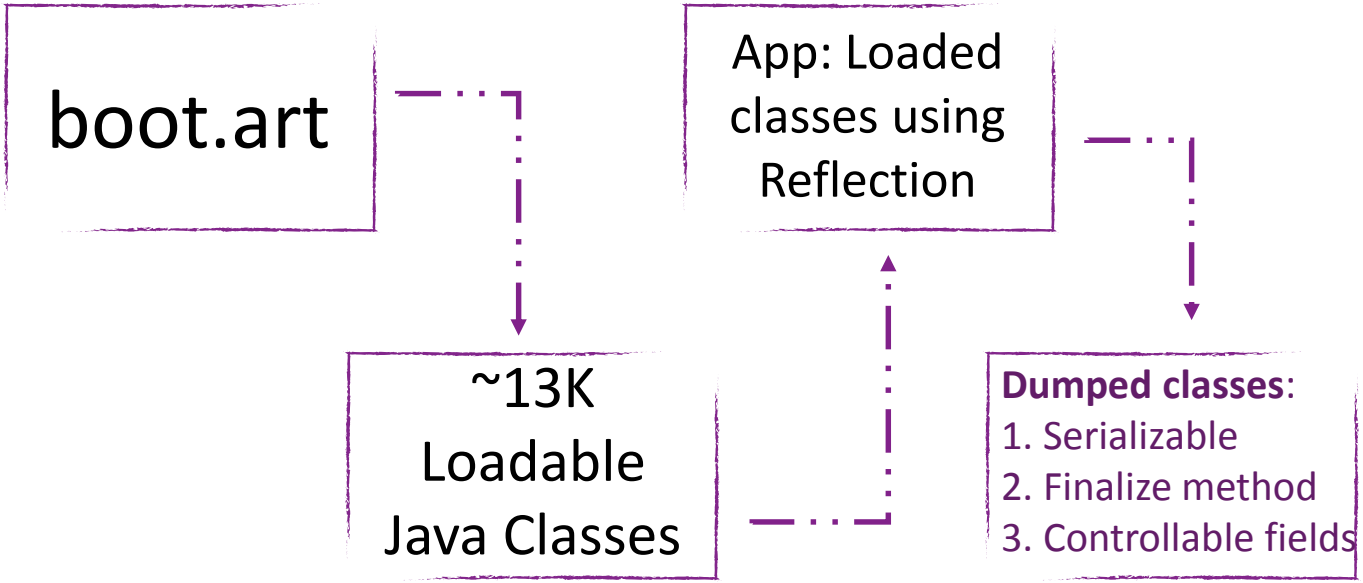
Experiment 1



Experiment 1



Experiment 1



The Result


```
OpenSSLX509Certificate
```

The Result


```
public class OpenSSLX509Certificate
extends X509Certificate {

  private final long mContext;

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

    NativeCrypto.X509_free(mContext);
  ...

  }
}
```

The Result


```
public class OpenSSLX509Certificate
extends X509Certificate {

  private final long mContext;

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

    NativeCrypto.X509_free(mContext);
  ...

  }
}
```

(1) SERIALIZABLE

The Result


```
public class OpenSSLX509Certificate
extends X509Certificate {

  private final long mContext;

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

    NativeCrypto.X509_free(mContext);
  ...

  }
}
```

(1) SERIALIZABLE

(2) CONTROLLABLE POINTER

The Result


```
public class OpenSSLX509Certificate
extends X509Certificate {

  private final long mContext;

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

    NativeCrypto.X509_free(mContext);
  ...

  }
}
```

(1) SERIALIZABLE

(2) CONTROLLABLE POINTER

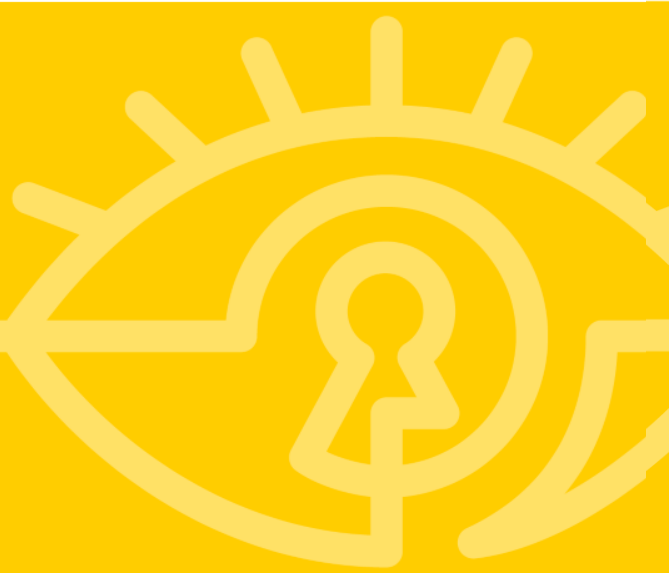
(3) EXECUTED AUTOMATICALLY BY THE GC

Arbitrary Decrement


```
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)
```

Arbitrary Decrement


```
ref = mContext+0x10
if (*ref > 0)
   *ref--
else
  free(…)
```



Proof-of-Concept Exploit

Arbitrary Code Execution in `system_server`

Exploit Outline

Exploit Outline

First Step of the Exploit

Own the Program Counter (PC)

Own the Program Counter

1. Override some offset / function ptr
2. Get it called.

Creating an Arbitrary Code Exec Exploit

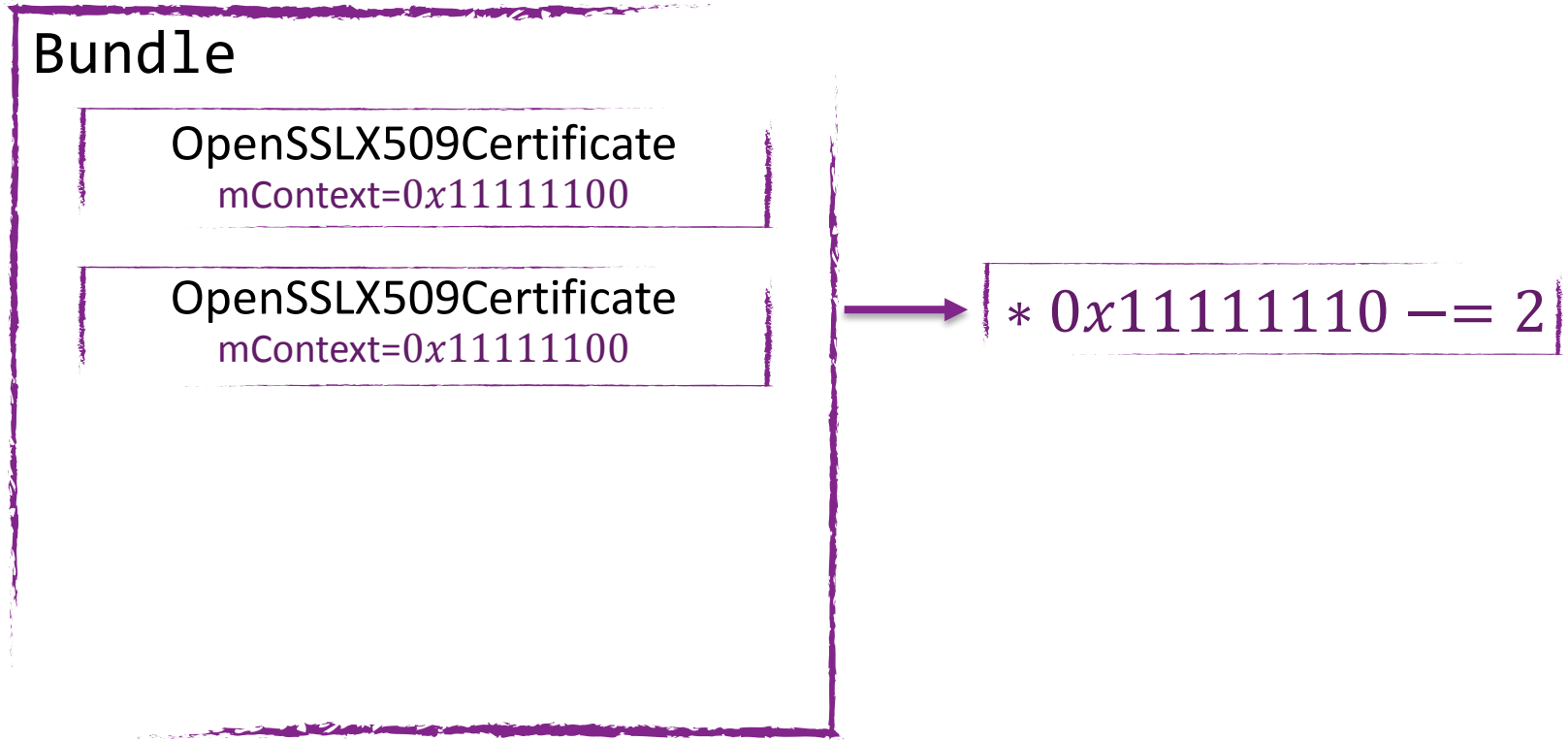
ARSENAL

1. Arbitrary Decrement
2. Controlled Buffer

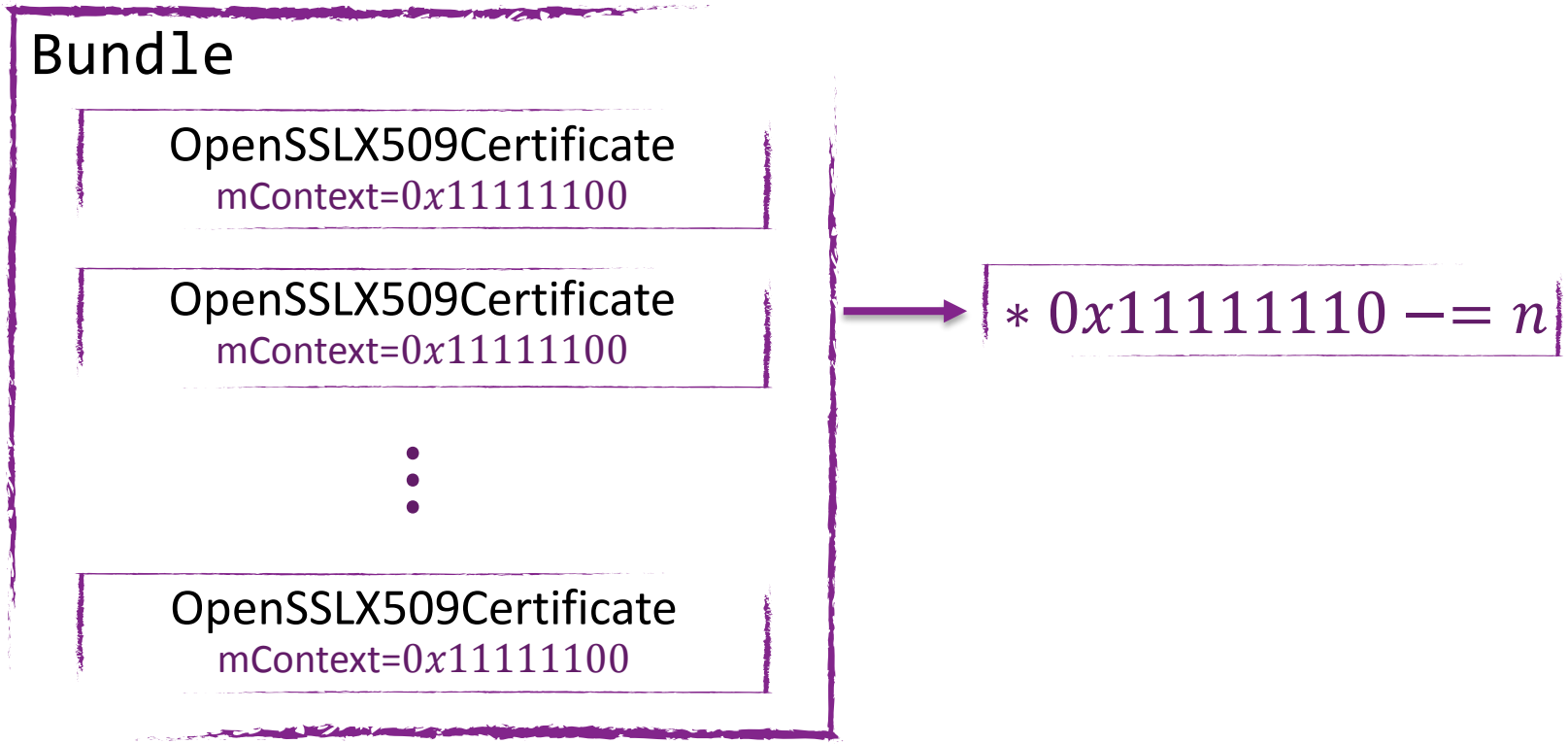
Constrained Arbitrary Memory Overwrite

$$*\,0x11111110 \mathrel{-}= 1$$

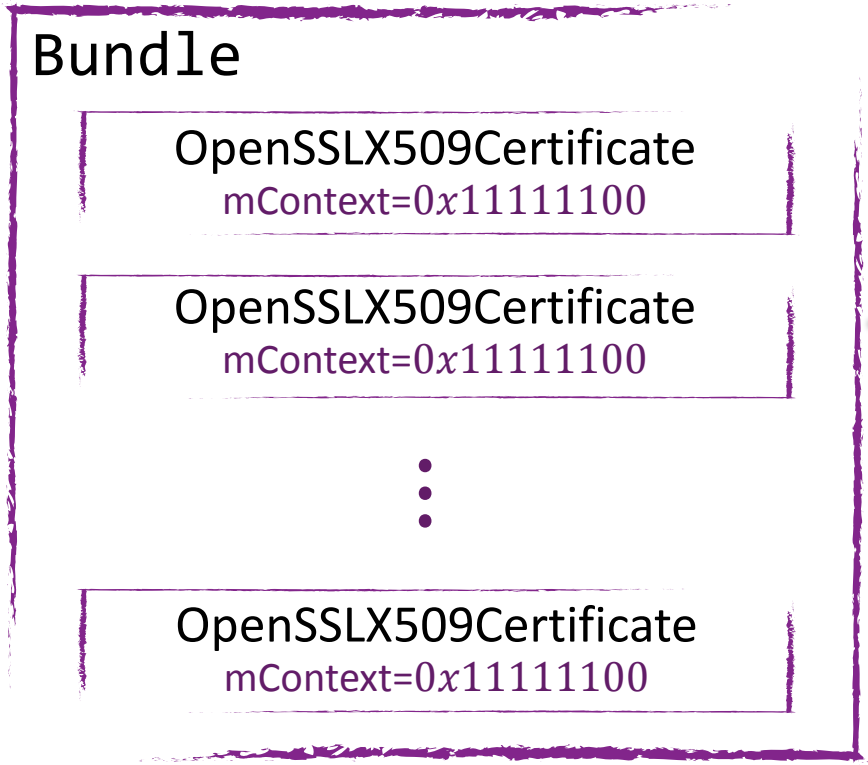
Constrained Arbitrary Memory Overwrite



Constrained Arbitrary Memory Overwrite



Constrained Arbitrary Memory Overwrite



$$*\,0x11111110 -= n$$

and If we knew the original value:

Arbitrary Overwrite

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

ARSENAL

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

   (if we knew the original value)

DEFENSES

1. ASLR
2. RELRO
3. NX pages
4. SELinux

Finding the original value: observation

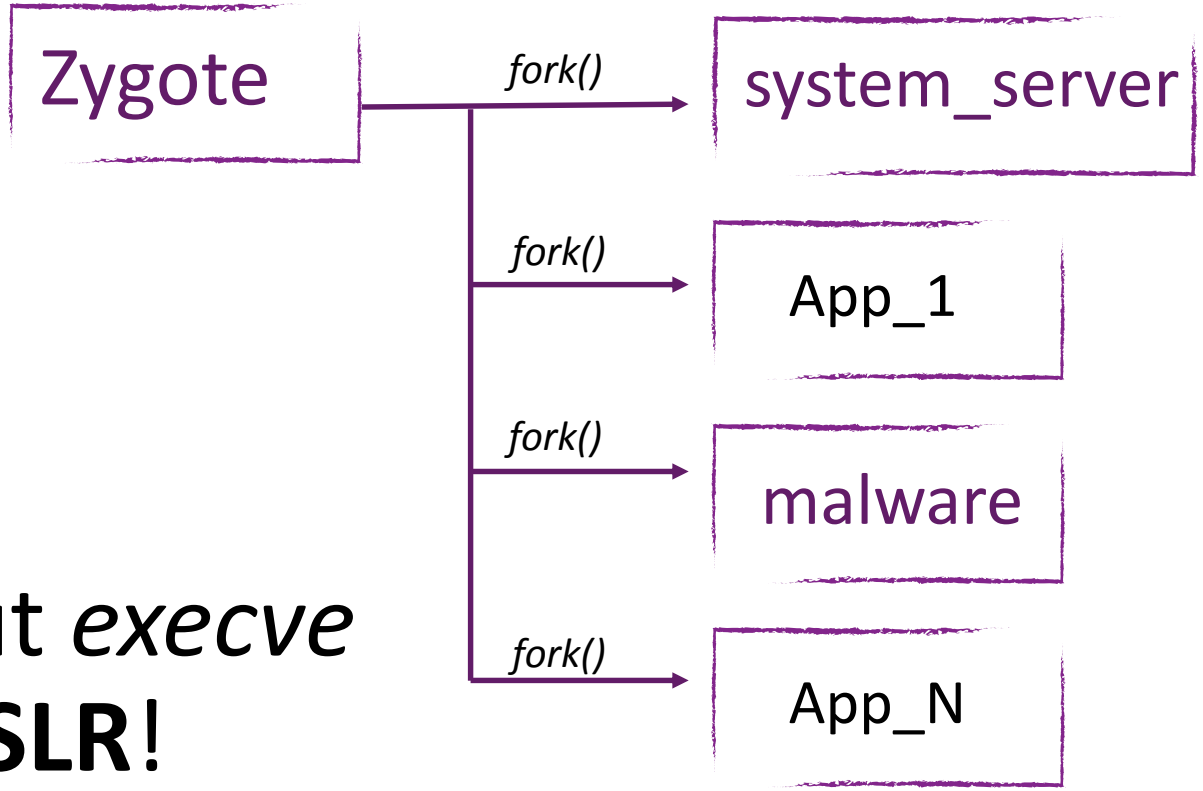
system_server

```
root@generic:/# cat /proc/<system_server>/maps

70e40000-72cee000 r–p  ...  boot.oat
72cee000-74400000 r-xp ...  boot.oat
74400000-74401000 rw-p ...  boot.oat
…
aa09f000-aa0c3000 r-xp ...  libjavacrypto.so
aa0c3000-aa0c4000 r--p ...  libjavacrypto.so
aa0c4000-aa0c5000 rw-p ...  libjavacrypto.so
…
b6645000-b66d5000 r-xp ...  libcrypto.so
b66d6000-b66e1000 r--p ...  libcrypto.so
b66e1000-b66e2000 rw-p ...  libcrypto.so
```

malware

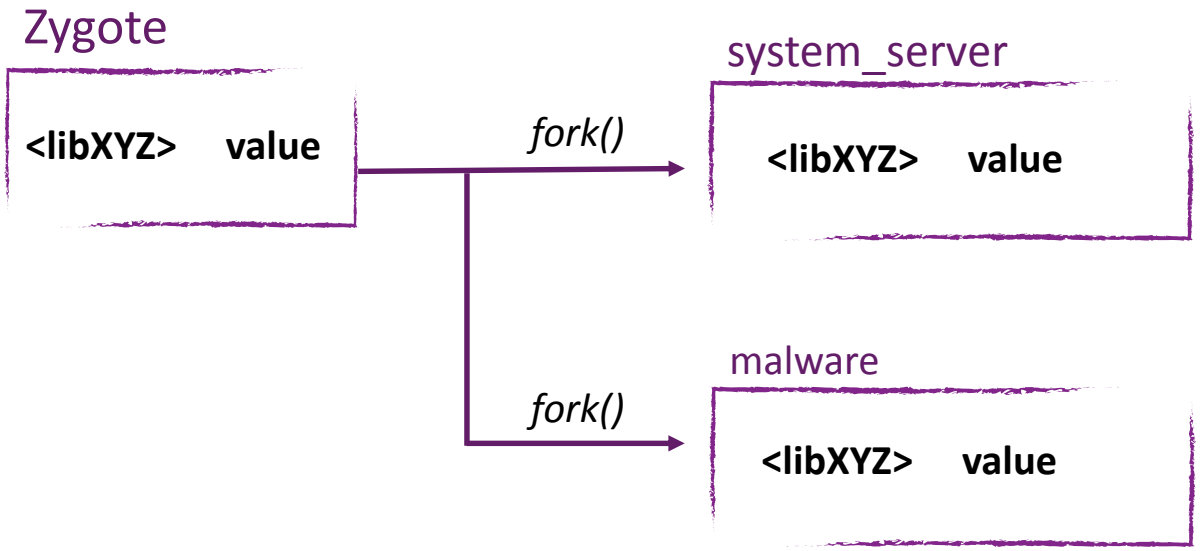
```
root@generic:/# cat /proc/<malware>/maps

70e40000-72cee000 r–p  ...  boot.oat
72cee000-74400000 r-xp ...  boot.oat
74400000-74401000 rw-p ...  boot.oat
…
aa09f000-aa0c3000 r-xp ...  libjavacrypto.so
aa0c3000-aa0c4000 r--p ...  libjavacrypto.so
aa0c4000-aa0c5000 rw-p ...  libjavacrypto.so
…
b6645000-b66d5000 r-xp ...  libcrypto.so
b66d6000-b66e1000 r--p ...  libcrypto.so
b66e1000-b66e2000 rw-p ...  libcrypto.so
```

Zygote app creation model



fork without *execve*
= no **ASLR**!

Determining the value



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. NX pages
4. SELinux

Using the Arbitrary Overwrite

Goal.

Overwite some pointer

Problem.

`.got` is read only (RELRO)

A Good Memory Overwrite Target

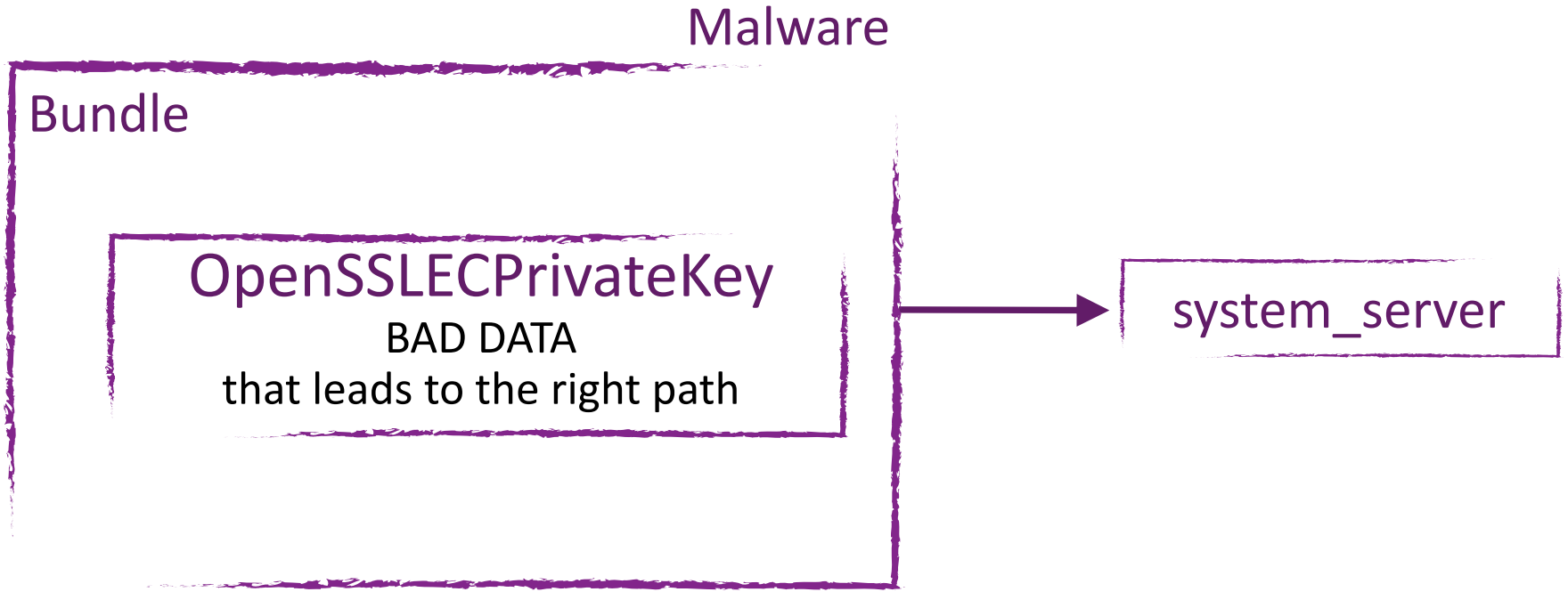
A function pointer under .data

`id_callback` in `libcrypto`

Called during deserialization of:

`OpenSSLECPrivateKey`

Triggering id_callback remotely



First Step Accomplished

We now own the Program Counter

Creating an Arbitrary Code Exec Exploit

ARSENAL

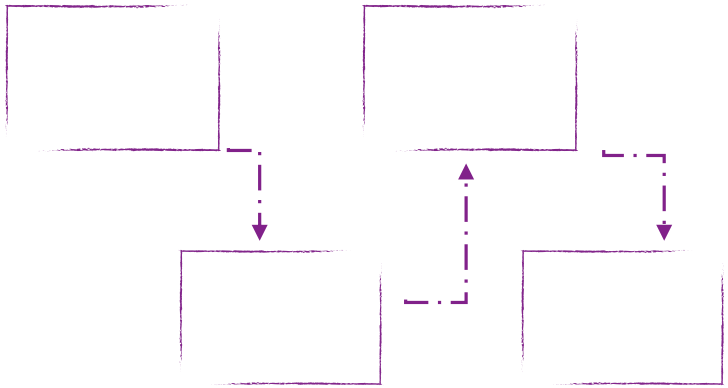
1. Arbitrary Decrement
2. Controlled Buffer
3. Arbitrary Overwrite

   (if we knew the original value)

DEFENSES

1. ~~ASLR~~
2. ~~RELRO~~
3. NX pages
4. SELinux

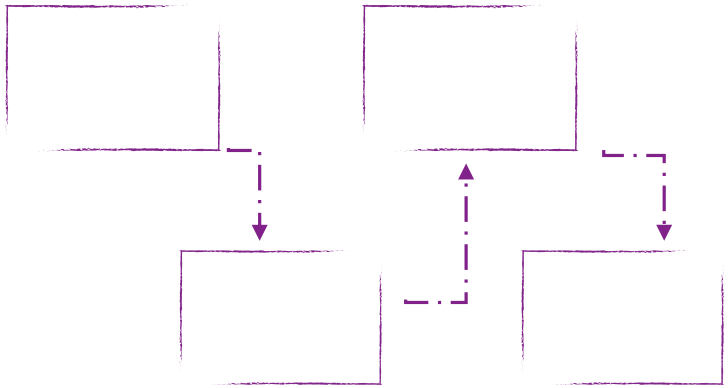
Next Steps of the PoC Exploit (simplified)



system_server

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

Problem 1: SP does not point at ROP chain

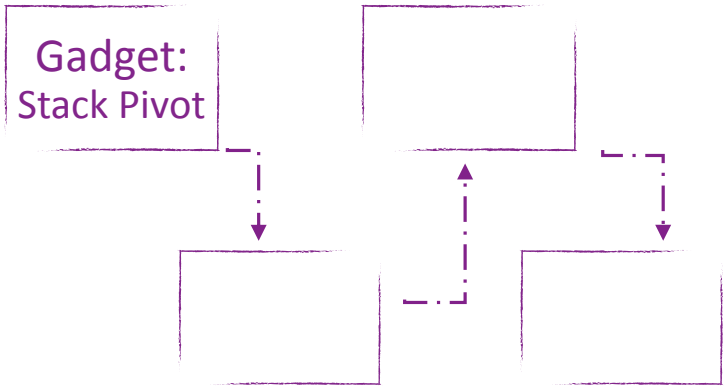


system_server

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 {…}

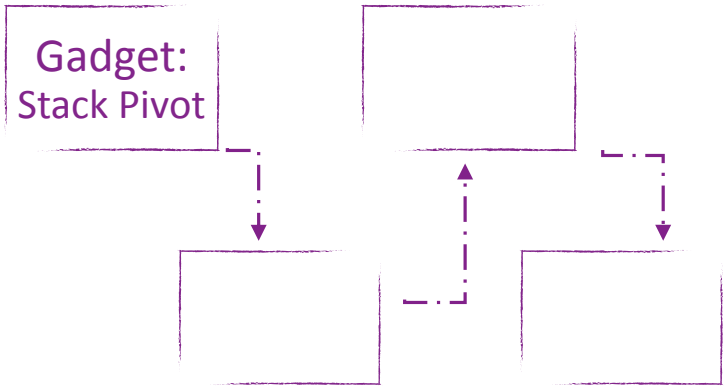


system_server

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

Solution: Stack Pivoting

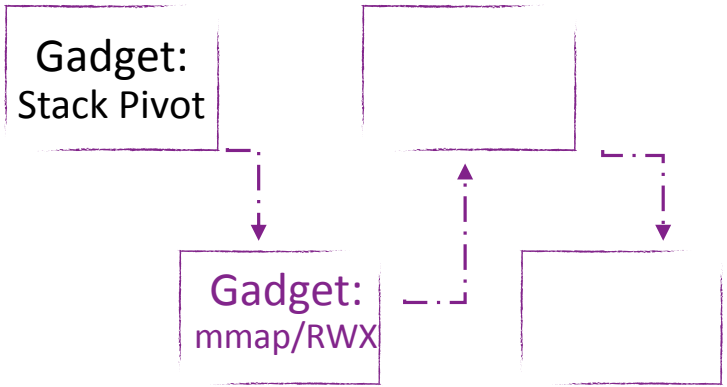
Our buffer happens to be pointed by fp.
The Gadget: mov sp, fp; …, pop {…}



system_server

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

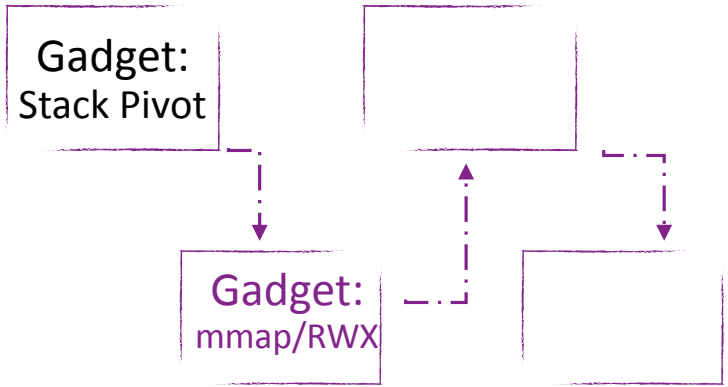
Allocating RWX Memory



system_server

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

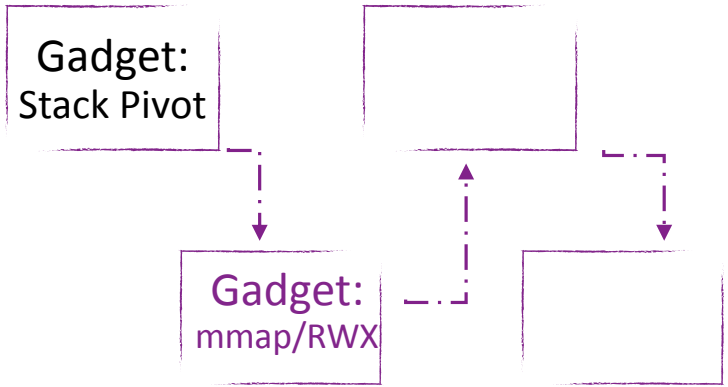
Problem 2: SELinux should prohibit mmap/RWX



system_server

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

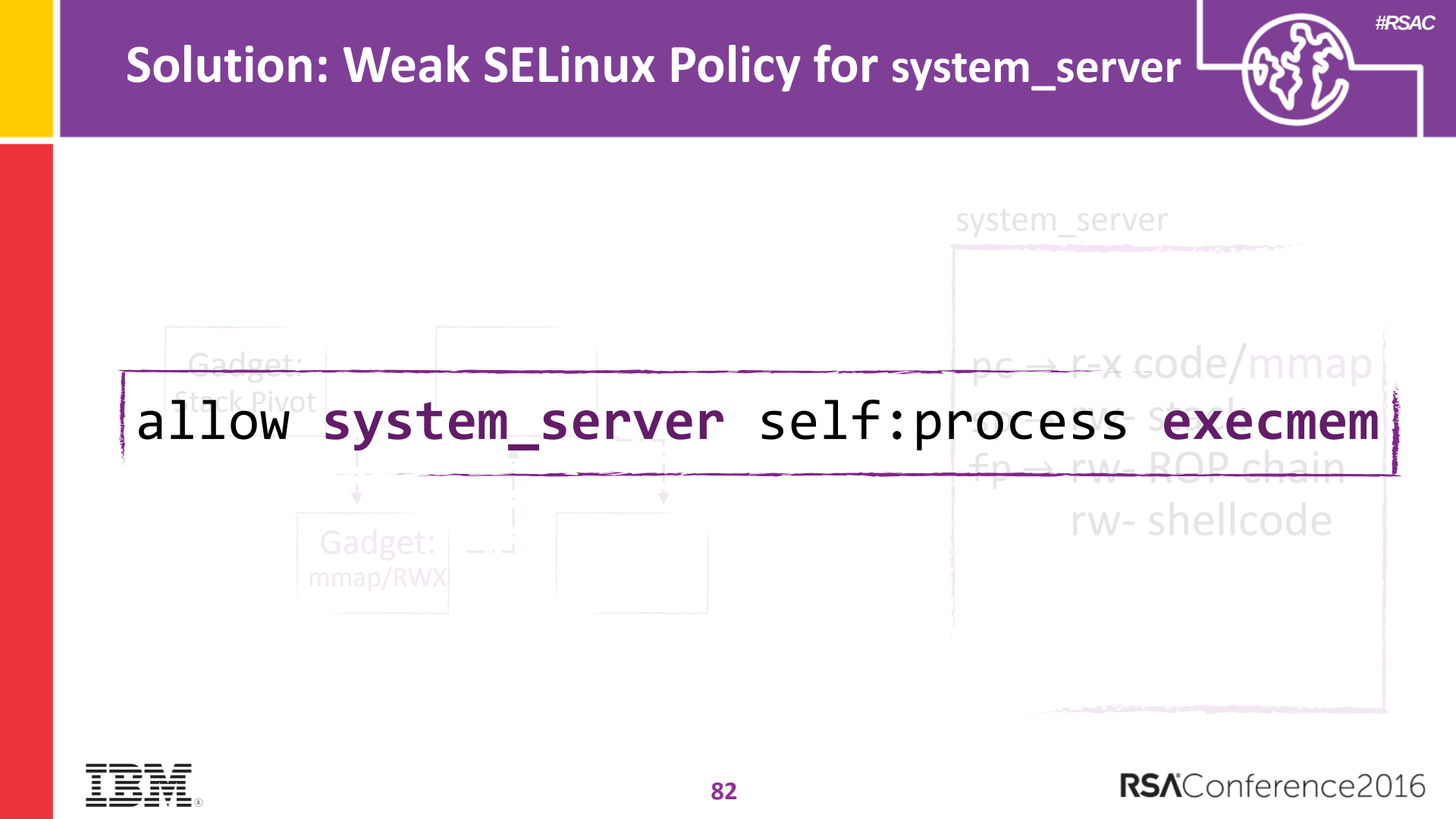
Solution: Weak SELinux Policy for system_server



system_server

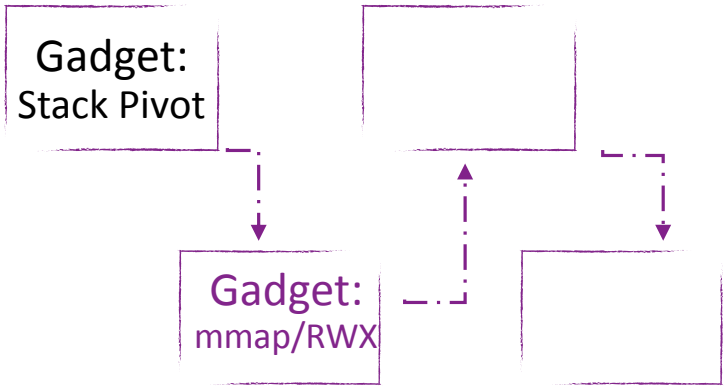
pc → r-x code/mmap
sp → rw- stack
fp → rw- ROP chain
rw- shellcode

Solution: Weak SELinux Policy for system_server




```
allow system_server self:process execmem
```

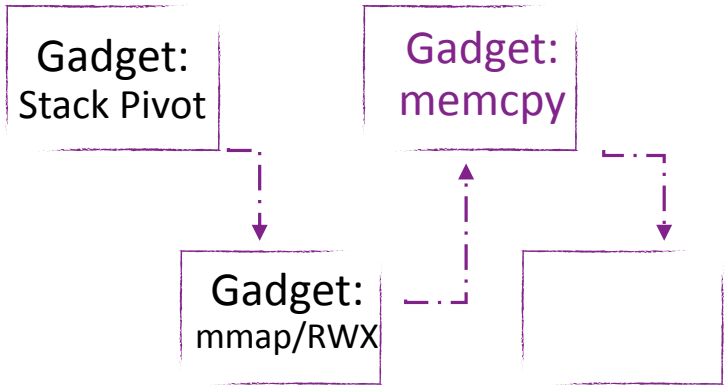
Allocating RWX Memory



system_server

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

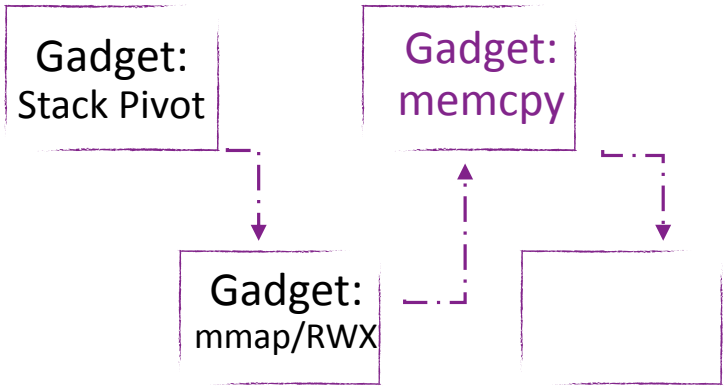
Copying our Shellcode



system_server

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

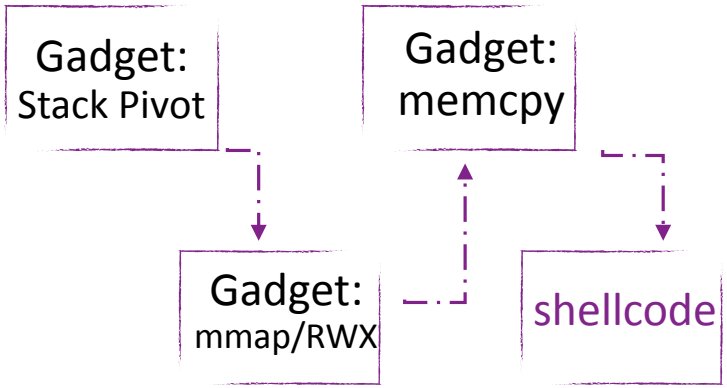
Copying our Shellcode



system_server

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

Executing our 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 knew the original value)

DEFENSES

1. ~~ASLR~~
2. ~~RELRO~~
3. ~~NX pages~~
4. ~~SELinux~~

Shellcode

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

- REPLACEMENT OF APPS
- SELINUX BYPASS
- ACCESS TO APPS' DATA
- KERNEL CODE EXEC (on select devices)

Demo

Google's Patch for CVE-2015-3825


```
public class OpenSSLX509Certificate
extends X509Certificate {

    private transient final long mContext;
    …

}
```

MISSING MODIFIER BEFORE OUR DISCLOSURE! (NOW PATCHED)

Hardened SELinux policy in the AOSP master branch

AOSP Commit #1:

android / platform / external / sepolicy / **23cde8776b94ff2228f3a8d845d4**

commit	23cde8776b94ff2228f3a8d845d41052af52319e	[log] [tgz]
author	Nick Kralevich <nnk@google.com>	Sat Aug 22 01:29:
committer	Nick Kralevich <nnk@google.com>	Sun Aug 23 21:34:
tree	e62b4bfb1abd09225b7cfa7af13fa170c206ad5b	
parent	acfd140c045d0bd295389a508ef6952acefb91fc [diff]	

```
system_server: remove old dalvik JIT rules on user/userdebug builds

On user and userdebug builds, system_server only loads executable
content from /data/dalvik_cache and /system. JITing for system_server
is only supported on eng builds. Remove the rules for user and
userdebug builds.

Going forward, the plan of record is that system_server will never
use JIT functionality, instead using dex2oat or interpreted mode.

Inspired by https://android-review.googlesource.com/98944

Change-Id: I5451 5acaae4792085869b89f0d21b87c66137510
```

AOSP Commit #2:

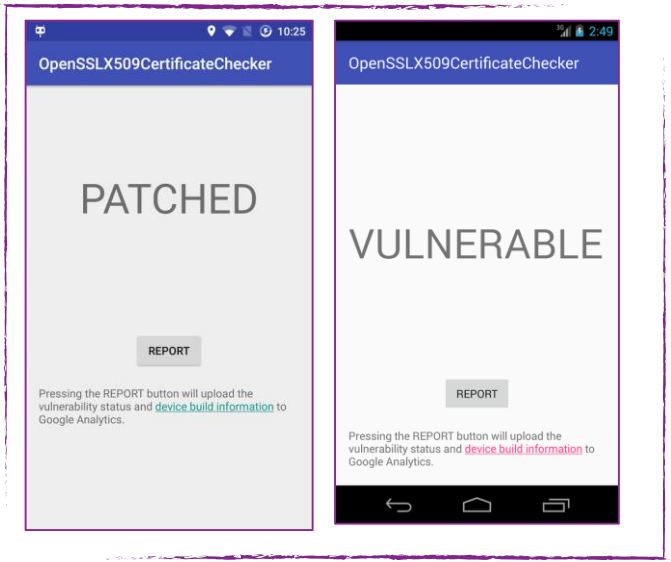
android / platform / external / sepolicy / **82bdd796e1265bd0e4b0497e9be**

commit	82bdd796e1265bd0e4b0497e9bed1d0cafc9883b	[log] [tgz]
author	Nick Kralevich <nnk@google.com>	Wed Oct 14 09:41
committer	Nick Kralevich <nnk@google.com>	Wed Oct 14 09:41
tree	0454c60c2b97adfb7ba0b196e4eeb60025219a6d	
parent	de11f5017c53aabba212425406962d21148fd2f6 [diff]	

```
system_server: (eng builds) remove JIT capabilities

23cde8776b94ff2228f3a8d845d41052af52319e removed JIT capabilities
from system_server for user and userdebug builds. Remove the capability
from eng builds to be consistent across build types.

Add a neverallow rule (compile time assertion + CTS test) to verify
this doesn't regress on our devices or partner devices.

Bug: 23468805
Bug: 24915206
Change-Id: Ib2154255c611b8812aa1092631a89bc59a27514b
```

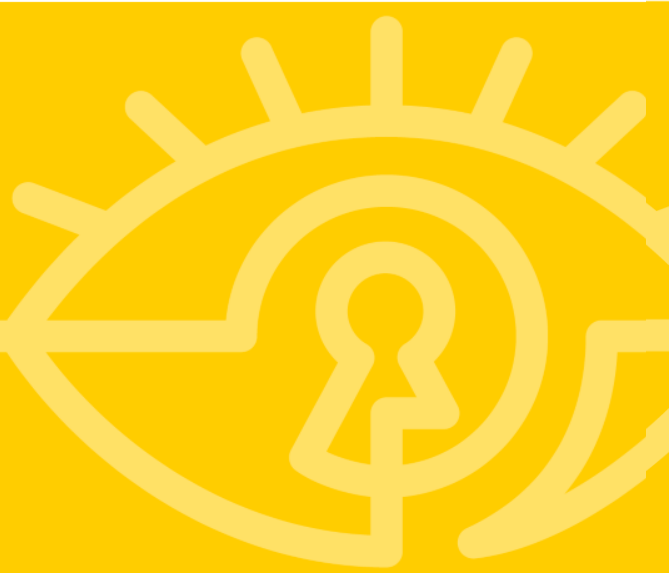
Are you patched?

Good news!
Majority of the devices are updated



GET IT ON Google Play

But some aren't…



Our 2nd Contribution: Vulnerabilities in SDKs

CVE-2015-2000/1/2/3/4/20

Finding Similar Vulnerabilities in SDKs

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

- Why. Fixing the Android Platform Vulnerability is not enough. Apps can be exploited as well!

Experiment 2

Analyzed over 32K of popular Android apps

Main Results

CVE-2015-2000	Jumio SDK	Code Exec.
CVE-2015-2001	MetaIO SDK	Code Exec.
CVE-2015-2002	Esri ArcGis SDK	Code Exec.
CVE-2015-2003	PJSIP PJSUA2 SDK	Code Exec.
CVE-2015-2004	GraceNote SDK	Code Exec.
CVE-2015-2020	MyScript SDK	Code Exec.

Root Cause (for most of the SDKs)

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

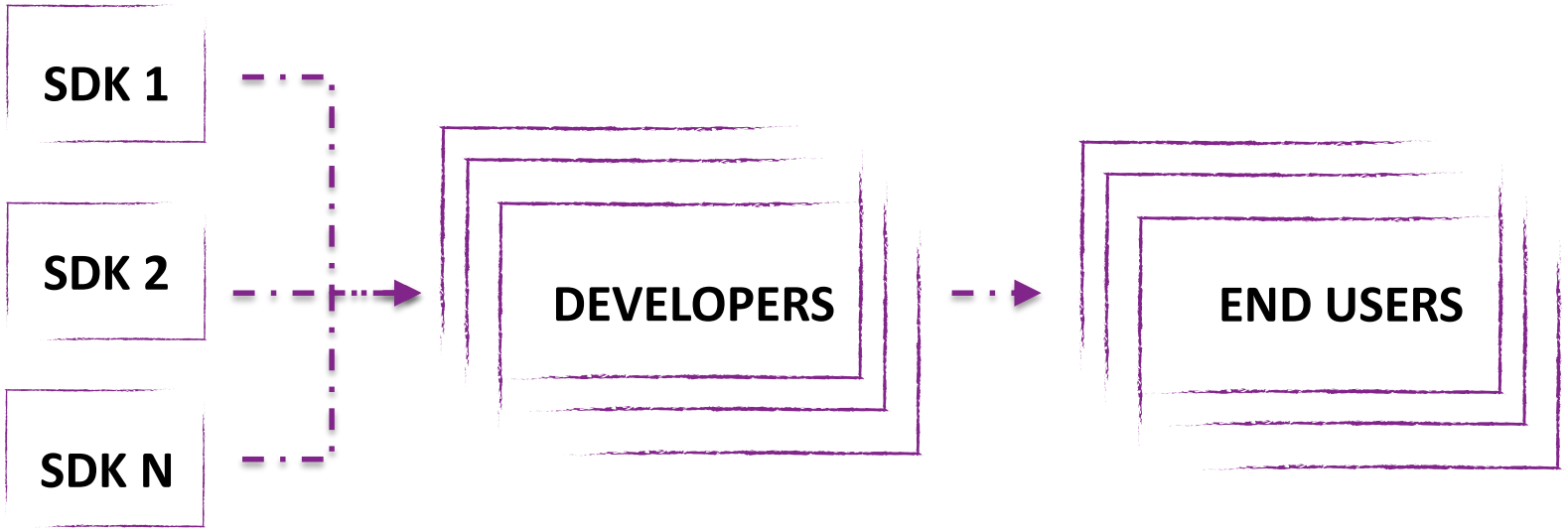
```
public class Foo implements Bar {
  private long swigCPtr;
  protected boolean swigCMemOwn;
  ...
  protected void finalize() {
     delete();
  }
  public synchronized void delete() {
    …
    exampleJNI.delete_Foo(swigCPtr);
    …
  }
  ...
}
```

POSSIBLY SERIALIZABLE

CONTROLLABLE POINTER

POINTER USED IN NATIVE CODE

Patching is problematic for SDKs



Apps are in a bad place

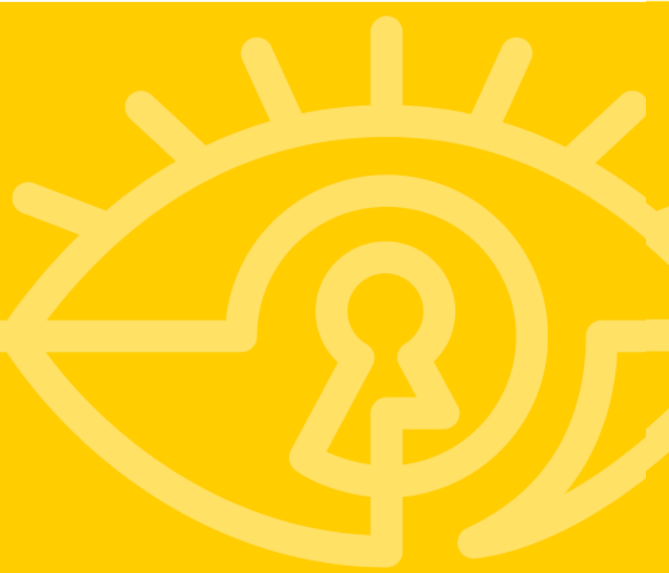
- **Vulnerable apps are still out there.**
  - SDKs need to be updated by app developers.
  - Dozens of apps still use them! (as of Feb ‘16)

- **New vulnerable apps can emerge**
  - Developers can introduce their own vulnerable classes.

Apps are in a bad place

- **Exploitation**
  - Still no **type-checking** by Android before deserialization.
  - **ASLR** can still be defeated when malware attacks Zygote forked processes.
  - As opposed to `system_server`, The **SELinux** policy hasn't been hardened for the **apps domain**.

```
10  # WebView and other application-specific JIT compilers
11  allow appdomain self:process execmem;
```



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.
  - **Consumers**: Update your Android.
  - **Developers**:
    - Update your SDKs.
    - Do not create vuln. Serializable classes. Use *transient* when needed!

Thank you!

?

ARE YOU STILL VULNERABLE?

GET IT ON Google Play

References

- **Paper.** https://www.usenix.org/conference/woot15/workshop-program/presentation/peles
- **Video.** https://www.youtube.com/watch?v=VekzwVdwqIY
- **Nexus Security Bulletin.** https://source.android.com/security/bulletin/2015-08-01.html
- **AOSP Patch.** https://android.googlesource.com/platform/external/conscrypt/+/edf7055461e2d7fa18de5196dca80896a56e3540
- **OpenSSLX509CertificateChecker.** https://play.google.com/store/apps/details?id=roeeh.conscryptchecker