MOGWAI LABS

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

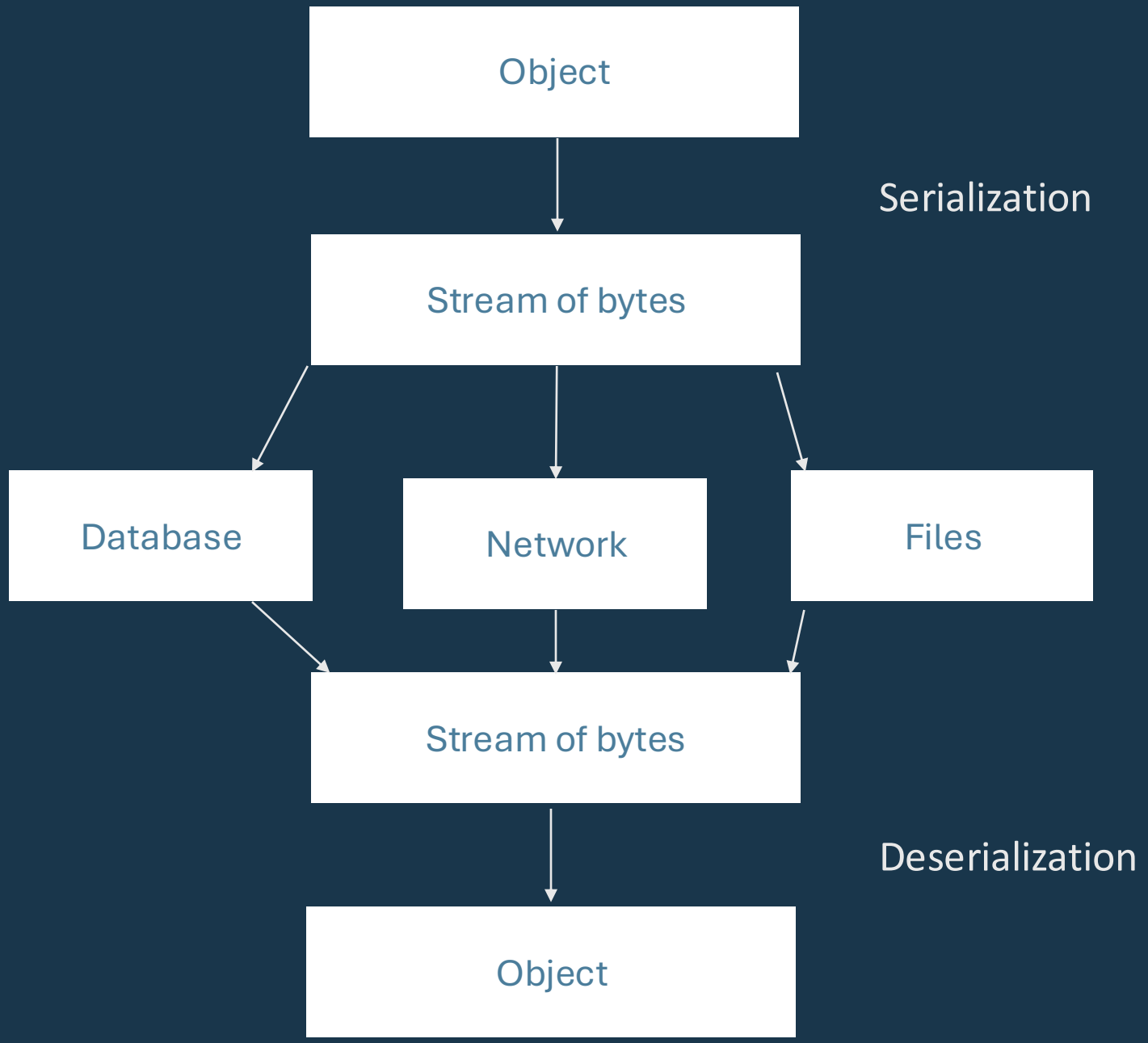
01 Deserialization Fundamentals

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 that 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

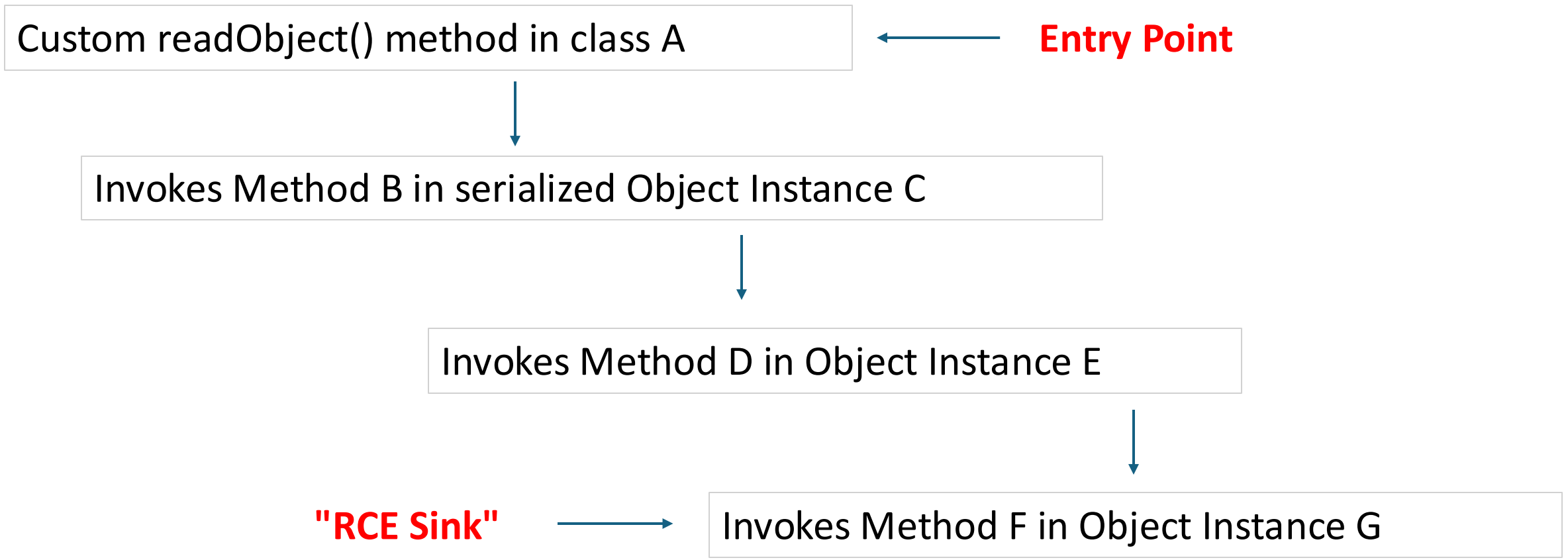
Here an example how to call a private method of a class.

```
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



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

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.getOuputProperties()`
- 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

MOGWAI LABS

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 to 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

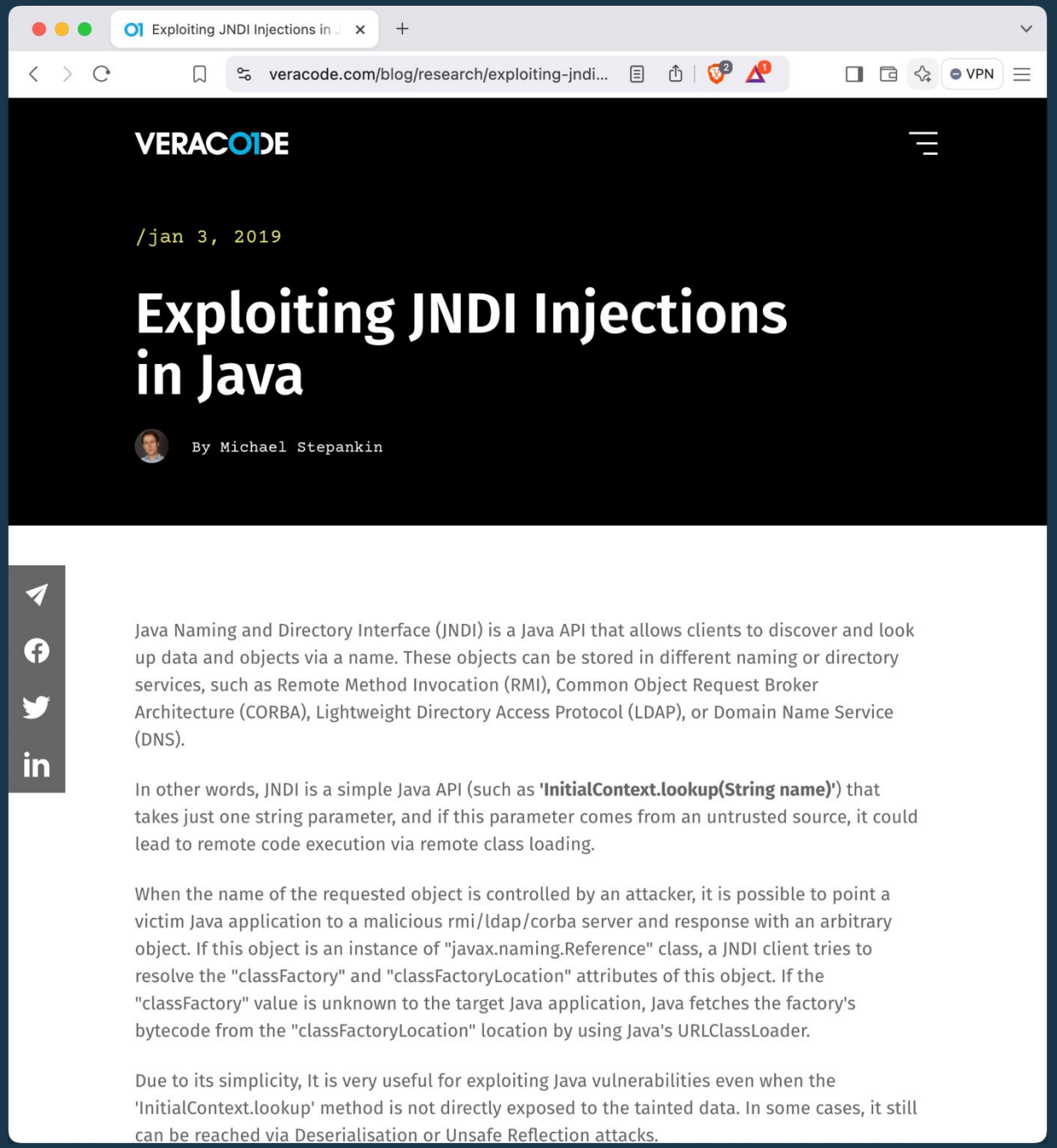
BeanFactory

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

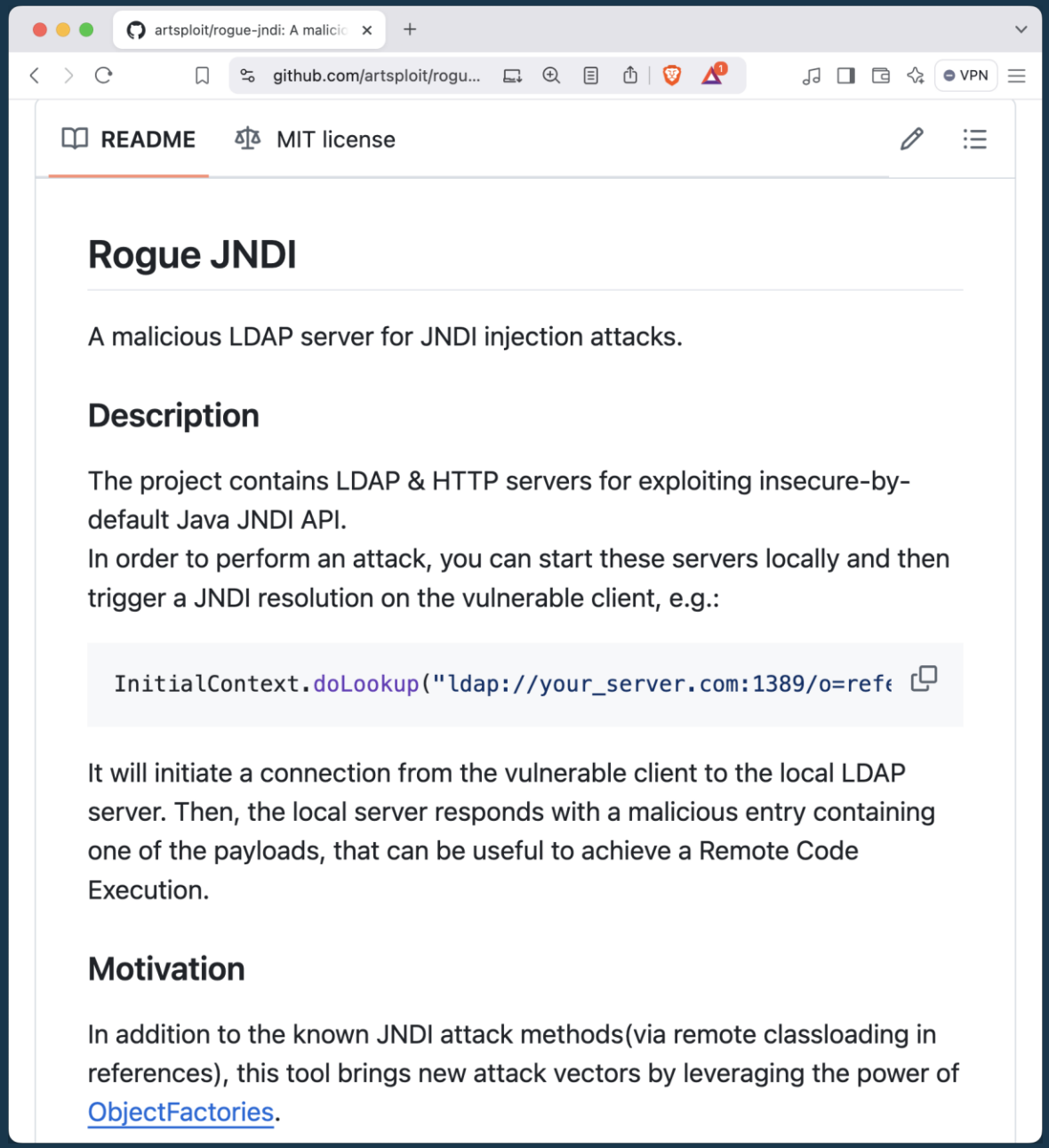


Rouge JNDI

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

Over time, some other ObjectFactories were added.

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



Patches

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

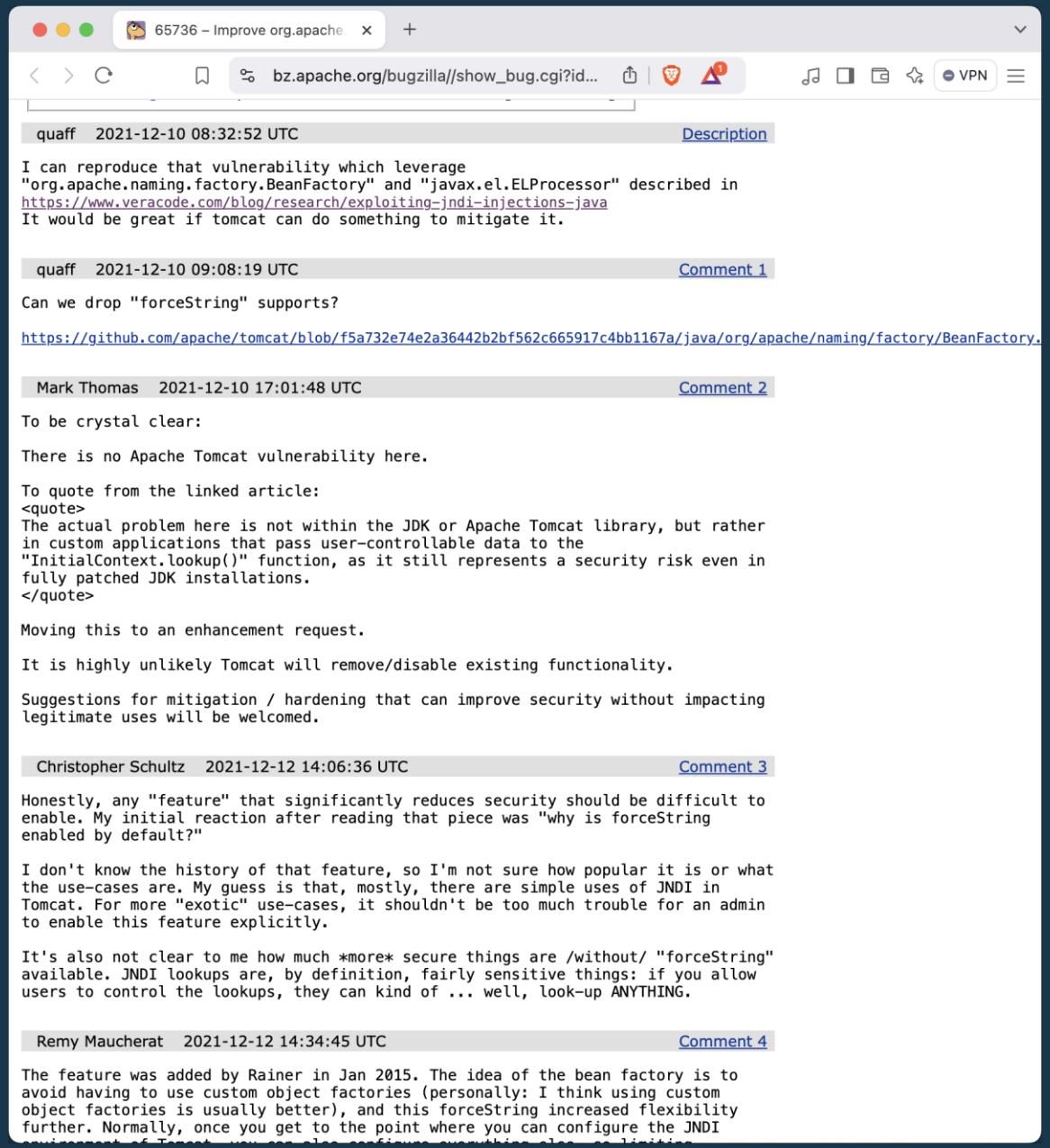
10.1.x for 10.1.0-M14 onwards

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

05 Exploitation in 2024

Get RCE or die trying

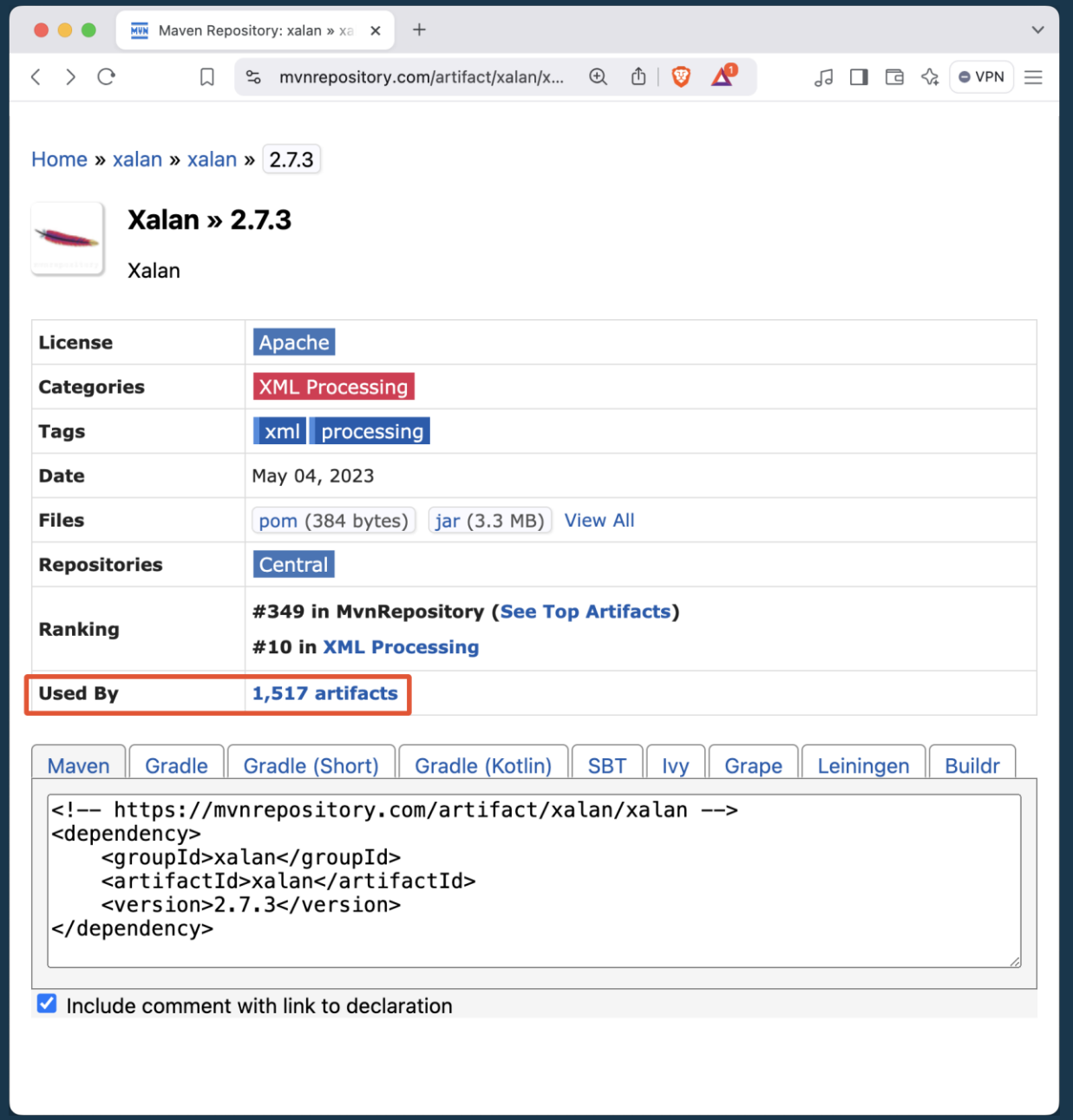
The possibility to invoke an arbitrary method is still a strong attack primitive! We just need other sinks!

Xalan-J

- The class `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl` 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 ☺



"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

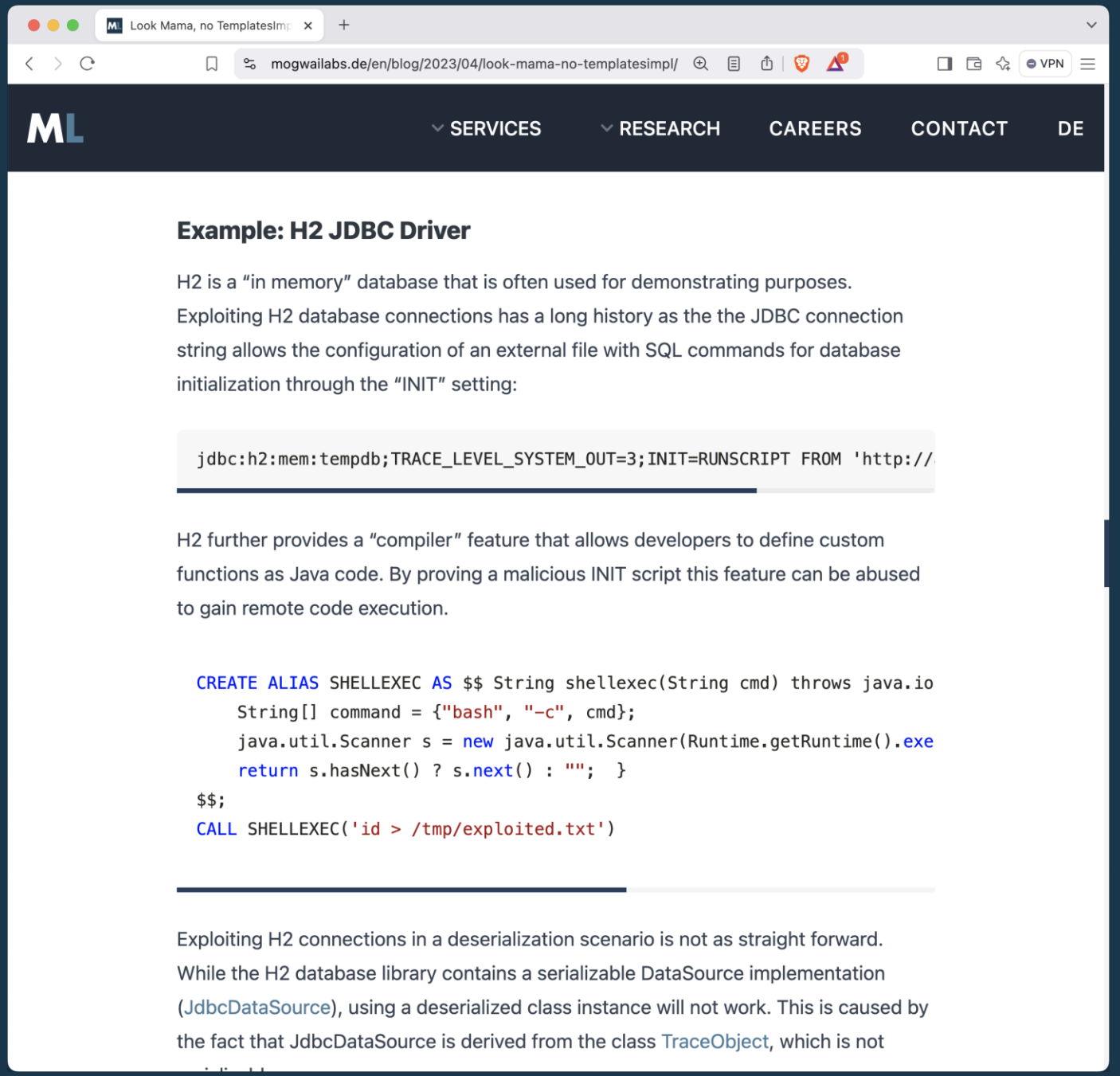
- Java allows you to set database connection properties in the connection string.
- JDBC drivers often provide a large attack surface
- Creating an outgoing connection to a database can often provide you RCE or arbitrary file read
- Can be archived through deserialization

RCE through H2

You can find a detailed writeup on our blog.

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

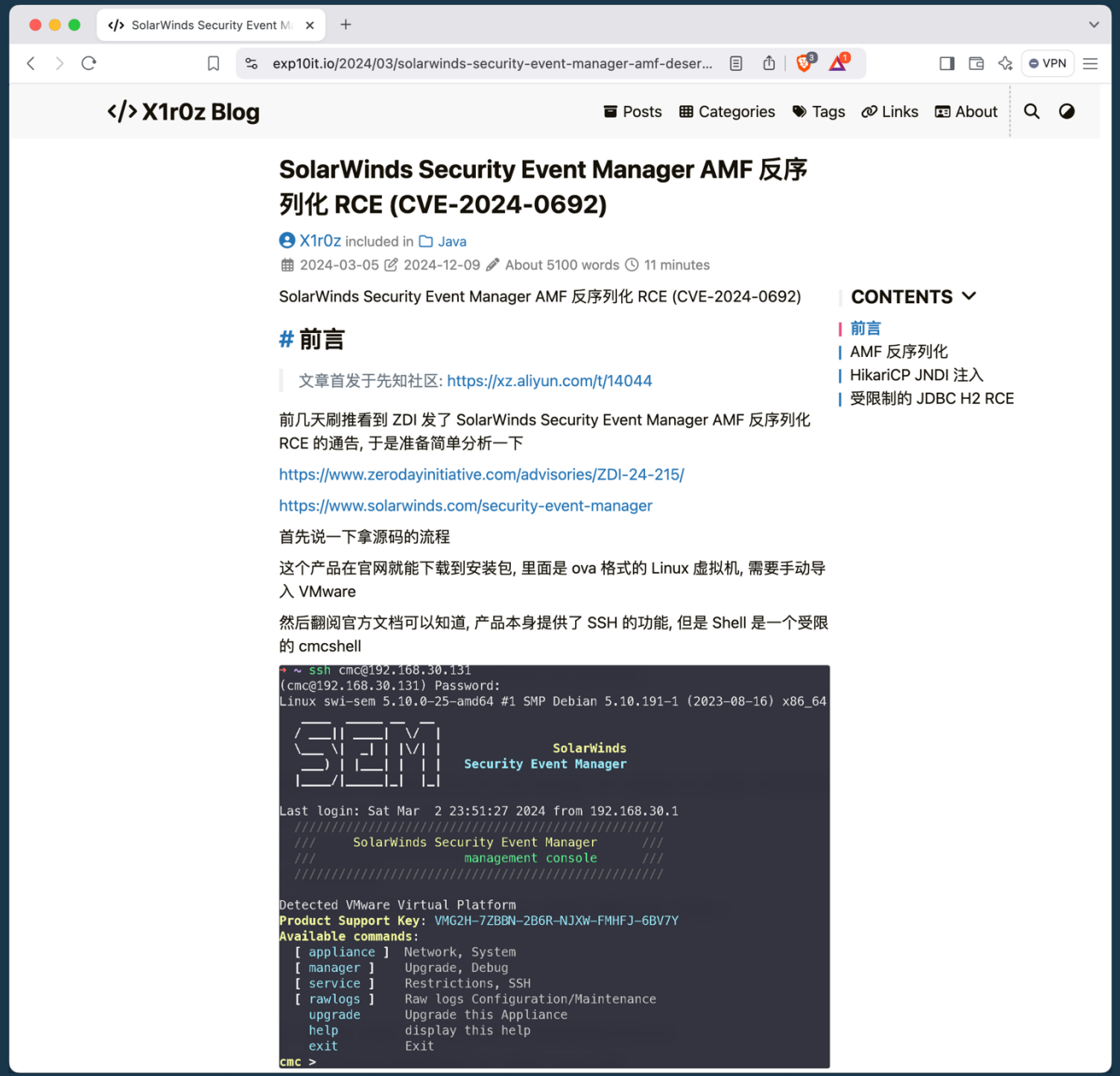
MOGWAI LABS



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/



MOGWAI LABS

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 Deserialzation (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