

Java安全漫谈 - 07.反序列化篇(1)

这是[代码审计知识星球](#)中Java安全的第七篇文章。

之前介绍了JMI的一些流程和攻击方法，我们在调试RMI时会发现，其发送、接收的数据都是反序列化数据。反序列化这个词在安全领域可谓红极一时，不同语言都拥有此类方法，且多少都拥有相关的漏洞。

那么，为什么反序列化常常会带来安全隐患？

一门成熟的语言，如果需要在网络上传递信息，通常会用到一些格式化数据，比如：

- JSON
- XML

JSON和XML是通用数据交互格式，通常用于不同语言、不同环境下数据的交互，比如前端的JavaScript通过JSON和后端服务通信、微信服务器通过XML和公众号服务器通信。但这两个数据格式都有一个共同的问题：不支持复杂的数据类型。

大多数处理方法中，JSON和XML支持的数据类型就是基本数据类型，整型、浮点型、字符串、布尔等，如果开发者希望在传输数据的时候直接传输一个对象，那么就不得不想办法扩展基础的JSON（XML）语法。

比如，Jackson和Fastjson这类序列化库，在JSON（XML）的基础上进行改造，通过特定的语法来传递对象；亦或者如RMI，直接使用Java等语言内置的序列化方法，将一个对象转换成一串二进制数据进行传输。

不管是Jackson、Fastjson还是编程语言内置的序列化方法，一旦涉及到序列化与反序列化数据，就可能会涉及到安全问题。但首先要理解的是，“反序列化漏洞”是对一类漏洞的泛指，而不是专指某种反序列化方法导致的漏洞，比如Jackson反序列化漏洞和Java `readObject`造成的反序列化漏洞就是完全不同的两种漏洞。

我们先来说说Java内置的序列化方法`readObject`，和其有关的漏洞。

反序列化方法的对比

在接触Java反序列化之前，相比大家多少都了解过其他语言的反序列化漏洞，其中极为经典的要数PHP和Python。

那么，Java的反序列化，究竟和PHP、Python的反序列化有什么异同？

Java的反序列化和PHP的反序列化其实有点类似，他们都只能将一个对象中的属性按照某种特定的格式生成一段数据流，在反序列化的时候再按照这个格式将属性拿回来，再赋值给新的对象。

但Java相对PHP序列化更深入的地方在于，其提供了更加高级、灵活的方法`writeObject`，允许开发者在序列化流中插入一些自定义数据，进而在反序列化的时候能够使用`readObject`进行读取。

当然，PHP中也提供了一个魔术方法叫`__wakeup`，在反序列化的时候进行触发。很多人会认为Java的`readObject`和PHP的`__wakeup`类似，但其实不全对，虽然都是在反序列化的时候触发，但他们解决的问题稍微有些差异。

Java设计`readObject`的思路和PHP的`__wakeup`不同点在于：`readObject`倾向于解决“反序列化时如何还原一个完整对象”这个问题，而PHP的`__wakeup`更倾向于解决“反序列化后如何初始化这个对象”的问题。

我知道这样说会比较难理解，也几乎没有文章说到和理解到这个微小的差异，但这个设计理念可以说是决定为什么Java的反序列化漏洞这么多的根本原因，我们这次用一整篇文章的篇幅来讨论一下这个问题。

PHP反序列化

PHP的序列化是开发者不能参与的，开发者调用 `serialize` 函数后，序列化的数据就已经完成了，你得到的是一个完整的对象，你并不能在序列化数据流里新增某一个内容，你如果想插入新的内容，只有将其保存在一个属性中。也就是说PHP的序列化、反序列化是一个纯内部的过程，而其 `__sleep`、`__wakeup` 魔术方法的目的就是在序列化、反序列化的前后执行一些操作。

一个非常典型的PHP序列化例子，就是含有资源类型的PHP类，如数据库连接：

```
1 <?php
2 class Connection
3 {
4     protected $link;
5     private $dsn, $username, $password;
6
7     public function __construct($dsn, $username, $password)
8     {
9         $this->dsn = $dsn;
10        $this->username = $username;
11        $this->password = $password;
12        $this->connect();
13    }
14
15    private function connect()
16    {
17        $this->link = new PDO($this->dsn, $this->username, $this->password);
18    }
19 }
```

PHP中，资源类型的对象默认是不会写入序列化数据中的。那么上述Connection类的 `$link` 属性在序列化后就是null，反序列化时拿到的也是null。

那么，如果我想要反序列化时拿到的 `$link` 就是一个数据库连接，我就需要编写 `__wakeup` 方法：

```
1 <?php
2 class Connection
3 {
4     protected $link;
5     private $dsn, $username, $password;
6
7     public function __construct($dsn, $username, $password)
8     {
9         $this->dsn = $dsn;
10        $this->username = $username;
11        $this->password = $password;
12        $this->connect();
13    }
14
15    private function connect()
```

```

16     {
17         $this->link = new PDO($this->dsn, $this->username, $this->password);
18     }
19
20     public function __sleep()
21     {
22         return array('dsn', 'username', 'password');
23     }
24
25     public function __wakeup()
26     {
27         $this->connect();
28     }

```

可见，这里 `__wakeup` 的工作就是在反序列化拿到 `Connection` 对象后，执行 `connect()` 函数，连接数据库。

`__wakeup` 的作用在反序列化后，执行一些初始化操作。但其实我们很少利用序列化数据传递资源类型的对象，而其他类型的对象，在反序列化的时候就已经赋予其值了。

所以你会发现，PHP的反序列化漏洞，很少是由 `__wakeup` 这个方法触发的，通常触发在析构函数 `__destruct` 里。其实大部分PHP反序列化漏洞，都并不是由反序列化导致的，只是通过反序列化可以控制对象的属性，进而在后续的代码中进行危险操作。

Java反序列化

Java反序列化的操作，很多是需要开发者深入参与的，所以你会发现大量的库会实现 `readObject`、`writeObject` 方法，这和PHP中 `__wakeup`、`__sleep` 很少使用是存在鲜明对比的。

我在《Java安全漫谈 - 06.RMI篇(3)》的最后一部分，讲到了 `ClassAnnotations`，这次再来说说 `ObjectAnnotation`。

Java在序列化时一个对象，将会调用这个对象中的 `writeObject` 方法，参数类型是 `ObjectOutputStream`，开发者可以将任何内容写入这个stream中；反序列化时，会调用 `readObject`，开发者也可以从中读取前面写入的内容，并进行处理。

举个例子，我编写了一个 `Person` 类：

```

1  package org.vulhub.Ser;
2
3  import java.io.IOException;
4
5  public class Person implements java.io.Serializable {
6      public String name;
7      public int age;
8
9      Person(String name, int age) {
10         this.name = name;
11         this.age = age;
12     }
13
14     private void writeObject(java.io.ObjectOutputStream s) throws
15         IOException {
16         s.defaultWriteObject();
17     }
18 }

```

```

16         s.writeObject("This is a object");
17     }
18
19     private void readObject(java.io.ObjectInputStream s)
20         throws IOException, ClassNotFoundException {
21         s.defaultReadObject();
22         String message = (String) s.readObject();
23         System.out.println(message);
24     }
25 }

```

可见，我这里在执行完默认的 `s.defaultWriteObject()` 后，我向stream里写入了一个字符串 `This is a object`。我们用上一章讲的工具 `SerializationDumper` 查看此时生成的序列化数据：

```

1  STREAM_MAGIC - 0xac ed
2  STREAM_VERSION - 0x00 05
3  Contents
4      TC_OBJECT - 0x73
5          TC_CLASSDESC - 0x72
6              className
7                  Length - 21 - 0x00 15
8                  Value - org.vulhub.Ser.Person -
9  0x6f72672e76756c6875622e5365722e506572736f6e
10     serialVersionUID - 0xf1 ad b2 c3 a9 83 d4 5c
11     newHandle 0x00 7e 00 00
12     classDescFlags - 0x03 - SC_WRITE_METHOD | SC_SERIALIZABLE
13     fieldCount - 3 - 0x00 03
14     Fields
15         0:
16             Int - I - 0x49
17             fieldName
18                 Length - 3 - 0x00 03
19                 Value - age - 0x616765
20         1:
21             Object - L - 0x4c
22             fieldName
23                 Length - 4 - 0x00 04
24                 Value - name - 0x6e616d65
25             className1
26                 TC_STRING - 0x74
27                 newHandle 0x00 7e 00 01
28                 Length - 18 - 0x00 12
29                 value -Ljava/lang/String; -
30  0x4c6a6176612f6c616e672f537472696e673b
31         2:
32             Object - L - 0x4c
33             fieldName
34                 Length - 8 - 0x00 08
35                 Value - password - 0x70617373776f7264
36             className1
37                 TC_REFERENCE - 0x71
38                 Handle - 8257537 - 0x00 7e 00 01
39             classAnnotations
40                 TC_ENDBLOCKDATA - 0x78
41                 superClassDesc
42                     TC_NULL - 0x70
43                     newHandle 0x00 7e 00 02

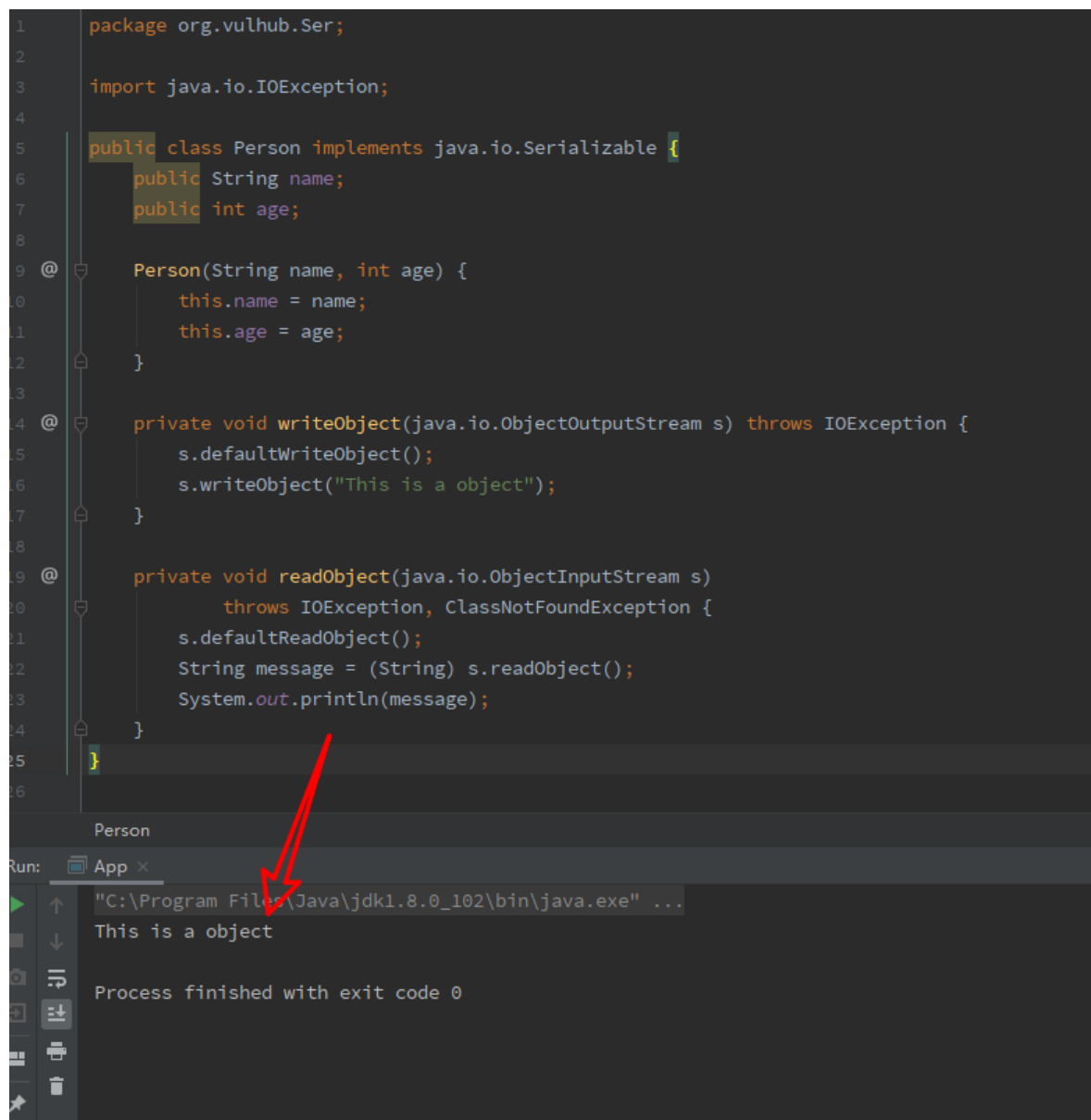
```

```
42 classdata
43     org.vulhub.Ser.Person
44     values
45         age
46             (int)22 - 0x00 00 00 16
47         name
48             (object)
49             TC_STRING - 0x74
50             newHandle 0x00 7e 00 03
51             Length - 3 - 0x00 03
52             Value - Bob - 0x426f62
53         password
54             (object)
55             TC_STRING - 0x74
56             newHandle 0x00 7e 00 04
57             Length - 6 - 0x00 06
58             Value - secret - 0x736563726574
59     objectAnnotation
60         TC_STRING - 0x74
61         newHandle 0x00 7e 00 05
62         Length - 16 - 0x00 10
63         Value - This is a object - 0x546869732069732061206f626a656374
64         TC_ENDBLOCKDATA - 0x78
```

可见，我们写入的字符串 `This is a object` 被放在 `objectAnnotation` 的位置。

在反序列化时，我读取了这个字符串，并将其输出：

```
1 package org.vulhub.Ser;
2
3 import java.io.IOException;
4
5 public class Person implements java.io.Serializable {
6     public String name;
7     public int age;
8
9     @
10     Person(String name, int age) {
11         this.name = name;
12         this.age = age;
13     }
14
15     @
16     private void writeObject(java.io.ObjectOutputStream s) throws IOException {
17         s.defaultWriteObject();
18         s.writeObject("This is a object");
19     }
20
21     @
22     private void readObject(java.io.ObjectInputStream s)
23         throws IOException, ClassNotFoundException {
24         s.defaultReadObject();
25         String message = (String) s.readObject();
26         System.out.println(message);
27     }
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



The screenshot shows an IDE with a Java file named `Person`. The code defines a `Person` class that implements `Serializable`. It has two fields, `name` and `age`, and three methods: a constructor, `writeObject`, and `readObject`. The `readObject` method reads a string from the input stream and prints it. The output window shows the command `"C:\Program Files\Java\jdk1.8.0_102\bin\java.exe" ...` and the output `This is a object`. The process finished with exit code 0.

这个特性就让Java的开发变得非常灵活。比如后面将会讲到的HashMap，其就是将Map中的所有键、值都存储在 `objectAnnotation` 中，而并不是某个具体属性里。

关于一些具体类是如何使用 `readObject` 方法的，我们后面在说到gadget的时候会详细分析。

Python反序列化

Python反序列化和Java、PHP有个显著的区别，就是Python的反序列化过程实际上是在执行一个基于栈的虚拟机。我们可以向栈上增、删对象，也可以执行一些指令，比如函数的执行等，甚至可以用这个虚拟机执行一个完整的应用程序。

所以，Python的反序列化可以立即导致任意函数、命令执行漏洞，与需要gadget的PHP和Java相比更加危险。

有关于Python反序列化的一些有趣的操作，可以参考我的另一篇文章《[Code-Breaking中的两个Python沙箱](#)》，我这里就不再赘述了。

总结一下，从危害上来看，Python的反序列化危害是最大的；从应用广度上来看，Java的反序列化是最常被用到的；从反序列化的原理上来看，PHP和Java是类似又不尽相同的。后文我将从一个非常简单的Gadget，即URLDNS入手，带大家深入理解反序列化漏洞的美妙。

