Who I Am

- Philippe Arteau
- Security Researcher at [GoSecure](https://gosecure.com)
- Open-source developer
  - Find Security Bugs (SpotBugs - Static Analysis for Java)
  - Security Code Scan (Roslyn – Static Analysis for .NET)
  - Burp and ZAP Plugins (Retire.js, CSP Auditor)

- Volunteer for the [nsec](https://nsecconf.com) conference and former trainer
Agenda

- Introduction
  - Deserialization
  - Gadget
- Exploitation
  - General methodology
  - Additional tricks
- History
  - Timeline of the discovery over the past 10 years
- Defense mechanisms
- Takeaways
Deserialization
“Serialization is the process of translating data structures or object states into a format that can be stored and reconstructed later in the same or another computer environment.”

[Ref : Wikipedia]
Deserialization Use Cases

System 1

Order
- order_id: D6C25D
- client_id: 42987
- items: [34, 68, 27]

System 2

Order
- order_id: D6C25D
- client_id: 42987
- items: [34, 68, 27]

- Storage
- Caching
- Inter-Process communication (Local)

- Network communication
- Message queue
How Objects Are Reconstructed

Depending on the implementation, the library or the function, it may:

▪ Initialized **fields**
▪ Call **Setters** (ie: setXXX or C# properties)
▪ Call **Constructor** with no arguments
▪ Call **custom hooks** intended to be called *specially on deserialization*
▪ Lifecycle methods : initialization, disposition (ie: __destruct in PHP), etc.

Libraries do their best to minimize side effects.
Exploitation Requirements

- Unsafe deserialization must be used
- A **gadget** allowing remote code execution must be available
- User-controlled data must be passed to a deserialization function
Simple Example

Unsafe deserialization

Gadget

```php
class sql_db {
    function __destruct() {
        $this->sql_close();
    }

    function sql_close() {
        [$...
        $this->createLog();
        [$...
    }

    function createLog() {
        $ip = $this->escape($_SERVER['REMOTE_ADDR']);
        $lang = $this->escape($_SERVER['HTTP_ACCEPT_LANGUAGE']);
        $agent = $this->escape($_SERVER['HTTP_USER_AGENT']);
        $log_table = $this->escape($this->log_table);
        $query = "INSERT INTO "$log_table." VALUES ('', '$ip', '$lang', '$agent');
        $this->sql_query($query);
    }
}

$result = unserialize($_GET['input'])
```
Specific Example: Java Native Serialization

```java
final ObjectInputStream objIn = new ObjectInputStream(in);
Command cmd = (Command) objIn.readObject();
```

- A class name is read from the bytestream
- The class is loaded from the name
- An object is instantiated from the class (no constructor is called)
- Custom `readObject()` is called if implemented
The Attack Surface

Entry point: *(The obvious part)*
- readObject()
- Setters/Getters
- Constructors

Trampoline methods: *(Not so obvious)*
- Java: hashcode(), equals(), Proxy and InvocationHandler
- .NET: Internal use of unsafe serializer (ie: BinaryFormatter)
- Ruby: Internal template evaluation
- PHP: Method name collision
Exploitation
General Method

1. Find serialized object in protocol
2. Generate a malicious payload with gadget X
3. Replace the initial object by the payload
   ▪ If it failed, generate a new malicious payload with a different gadget
   ▪ If it failed, transform the existing Object stream

If it still does not work, the classes might not be available or allowed (white or blacklist)
Demonstration

ysoserial.net used to generate a payload for a ASP.net application
Detection with DNS (Java)

https://www.gosecure.net/blog/2017/03/22/detecting-deserialization-bugs-with-dns-exfiltration
https://blog.paranoidsoftware.com/triggering-a-dns-lookup-using-java-deserialization/
How to Generate “DNS” Payload Using Ysoserial

Example:

$ java -jar ysoserial-0.0.5-all.jar URLDNS http://8pygg0brnl4ofg3spss6l17q1h77vw.burpcollaborator.net > payload.bin

- **URLDNS**: Gadget
- **http://8pygg0brnl4ofg3spss6l17q1h77vw.burpcollaborator.net**: URL that will be resolved.
A new deserialization vector was found in PHP recently.

- It concerns user input being passed to:
  - `fopen()`
  - `copy()`
  - `file_exists()`
  - `filesize()`

```
file_exists("phar://userfile.bin")
```

The metadata from the **PHP Archive (PHAR)** is serialized

History of Deserialization
First Deserialization Vulnerability (CWE-502)

- CVE-2007-1701 (PHP 4.4.6)

- Double free vulnerability was found in `session_decode`
- The vulnerability can be triggered if `register_globals` is enabled or if the application bypasses user content to the function directly

- While it affects a deserialization function, it is not representative of the most common deserialization vulnerabilities.
First “Gadget Based” Vulnerability

- CVE-2011-2894
- Spring vulnerability discovered by Wouter Coekaerts

- One of the first “gadget-based” vulnerabilities
- The Spring team mitigate both:
  - The unrestricted deserialization
  - The gadget
- It use a common pattern – *Proxy* + *InvocationHandler* – that will be reused in most of the Java gadgets.

Important dates in Java Deserialization History

2011
First Java deserialization vulnerability leading to RCE found in Spring

2013
Look-Ahead Class Validation article By Pierre Ernst

2015 Jan.
Presentation at AppSecCal 2015 about the potential of the Apache Commons Collection Gadget

2015 Nov.
POCs are published by Foxgloves Security for multiples enterprise applications

2016
YSoSerial – a tool that generate gadget – now has 29 different gadgets

2017
Paper publish on deserialization vulnerability in YAML, JSON and AMF parser.

2018
Deserialization Filtering introduce in JDK 9 (Requires config.)

Ref: All the articles are in the references section
Gadgets timeline in Ruby, Java, .NET and PHP

2013
First Ruby gadget targeting specifically ActiveSupport from RubyOnRails

2011
First Java gadget found that could be used to leverage a RCE in Spring applications (Wouter Coekaerts)

2015
Initial version of ysoserial
Targeting Java native serialization with Commons-Collection Gadget (Chris Frohoff)

2017
MarshalSec is released.
Targeting various JSON and XML Java parsers. (Moritz Bechler)

2017
PHP GGC is released with gadgets in many popular PHP frameworks (ambionics team)

2017
ysoserial.net is released.
Targeting JSON.net, BinaryFormatter and others parser. (Alvaro Muñoz)

2018
Universal Gadget for Ruby with no specific gem needed (Luke Jahnke)

Ref: All the articles are in the references section
CWE-502: Deserialization of Untrusted Data

Dataset taken from: https://www.cvedetails.com/vulnerability-list/cweid-502/vulnerabilities.html
What Will Happen Next?

▪ Some gadgets will stop working eventually
▪ No gadgets are found **yet** in some platforms:
  ▪ .NET Core
  ▪ .NET on Linux (With no 3rd party library)
  ▪ Universal PHP gadget
  ▪ PHP gadget for WordPress
▪ Frameworks and libraries will likely start to blacklist common classes from deserialization (when possible).
Defense Mechanisms
Using Safe Libraries (not error-prone)

- Not all libraries are created equal
- Some libraries have strict class validation during deserialization
- Refer to paper: **Friday the 13th JSON attacks (BH2017)**
### Using Safe(r) Libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Type Discriminator</th>
<th>Type Control</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast.JSON</td>
<td>.NET</td>
<td>Default</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>Json.Net</td>
<td>.NET</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection (weak)</td>
<td>Setter, Deser., Callbacks, Type Converters</td>
</tr>
<tr>
<td>FSPickler</td>
<td>.NET</td>
<td>Default</td>
<td>Expected Object Graph Inspection (weak)</td>
<td>Setter, Deser., callbacks</td>
</tr>
<tr>
<td>Sweet.Jayson</td>
<td>.NET</td>
<td>Default</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>JavascriptSerializer</td>
<td>.NET</td>
<td>Configuration</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>DataContract JsonSerializer</td>
<td>.NET</td>
<td>Default</td>
<td>Expected Object Graph Inspection (strong)</td>
<td>Setter, Deser., callbacks</td>
</tr>
<tr>
<td>Jackson</td>
<td>Java</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection (weak)</td>
<td>Setter</td>
</tr>
<tr>
<td>Genson</td>
<td>Java</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection (weak)</td>
<td>Setter</td>
</tr>
<tr>
<td>JSON.IO</td>
<td>Java</td>
<td>Default</td>
<td>Cast</td>
<td>toString</td>
</tr>
</tbody>
</table>

- Some libraries are less error-prone
- Deserialization with user-input should **at least have graph inspection**

Taken from *Friday the 13th JSON attacks* paper [https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf](https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf)
Use Blacklist or Whitelist Mechanisms

- Libraries may contain configurable whitelist and blacklist
  - Xstream (Java): allowTypeHierarchy, allowTypesByRegExp, ...
  - JSON.net (C#): ContractResolver

- 3rd party libraries could be used to accommodate
  - NotSoSerial, contrast-rO0, commons-io (class ValidatingObjectInputStream)

Some vendors – namely Weblogic – have chosen to use blacklist[1]

Takeaways

- **Attack tools** only get better
- **Frameworks and libraries** also do get better
- Prefer libraries with built-in class validation
- Deserialization is a complex attack vector
  - Gadgets can take quite some time to be discovered
  - Once discover the exploitation becomes trivial
Questions?

Contact

parteau@gosecure.ca
https://gosecure.net/
@h3xStream @GoSecure_Inc
References
What Do WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, and Your Application Have in Common? by Stephen Breen

AppSecCali 2015 - Marshalling Pickles by Christopher Frohoff and Gabriel Lawrence

Exploiting Deserialization Vulnerabilities in Java by Matthias Kaiser

Java Serialization Cheat-Sheet

YSoSerial tool maintained by Christopher Frohoff

Look-ahead Java deserialization by Pierre Ernst

NotSoSerial java-agent for mitigation
- **hack.lu CTF challenge 21 writeup**: Simple example with PHP `unserialize`
- **PHP magic methods**
- **PHP GGC**
Ruby References

- First Ruby gadget [http://phrack.org/issues/69/12.html](http://phrack.org/issues/69/12.html)
▪ Ysoserial.net : Payload generator

https://github.com/pwntester/ysoserial.net

▪ Friday The 13th JSON Attack - White Paper


▪ New attack vector in .NET

https://illuminopi.com/assets/files/BSidesIowa_RCEvil.net_20190420.pdf