Deserialization Vulnerabilities

Jaroslav Lobačevski

New item in OWASP Top 10

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Injection	→	A1:2017 – Injection
A2 – Broken Authentication and Session Management	→	A2:2017 – Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2013 – Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017 – XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017 – Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017 – Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017 – Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017 – Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017 – Insufficient Logging & Monitoring [NEW, Comm.]

In the News

16 Dec 2013 on Java | Exploit | CVE-2011-2894 | Spring

CVE-2011-2894: Deserialization Spring RCE

This post is about an old RCE vulnerability in applications deserializing streams from untrusted sources and hav

PayPal Engineering

Lessons Learned from the Java Deserialization Bug

By Laksh Raghavan January 21, 2016

Severe Deserialization Issues Also Affect .NET, Not Just Java

By Catalin Cimpanu August 7, 2017 10:58 AM 0

The .NET ecosystem is affected by a similar flaw that has wreaked havoc among Java apps and developers in 2016.

duce and eliminate
able proactive security

Zero though

Text January 22, 2016

Spring framework deserialization RCE

Spring framework is commonly used 3rd party library used by projects. If spring-tx.jar, spring-commons.jar and javax.trans

.NET deserialization flaw found in popular .NET projects

To show that the flaw they discovered can affect real-world apps, and is not just a theoretical threat, researchers identified: CVE-2017-9424 — a JSON deserialization flaw in Breeze, a .NET data management backend framework; and CVE-2017-9785 — a JSON deserialization flaw in NancyFX, a lightweight .NET web framework based on Ruby's Sinatra.

Some Facts

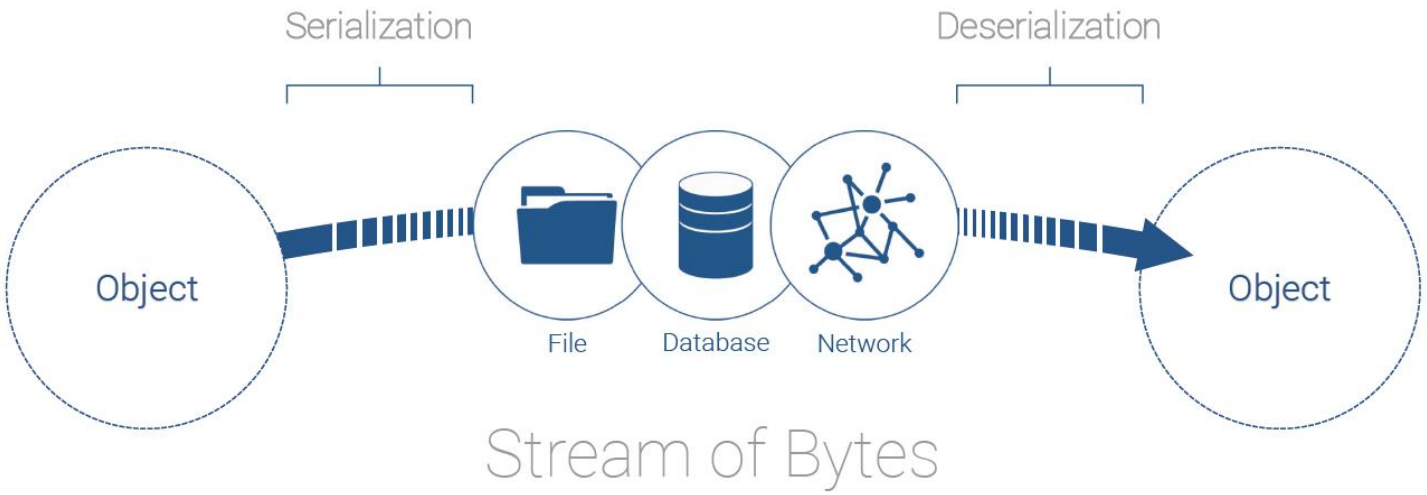
- Known attack vector since 2011
- Previous lack of good RCE gadgets
- Gadgets' discovery caught many off-guard
  - 2015 Chris Frohoff & Gabriel Lawrence RCE gadget in the Apache CommonsCollections library
  - 2016 Florian Gaultier - First possibility of a .NET RCE gadget
  - 2017 James Forshaw – two .NET RCE gadgets in three known formatters
  - 2017 Alvaro Muñoz & Oleksandr Mirosh more gadgets in .NET & Java

2018. Is it over yet?

- YamlDotNet deserialization security update in v5.0.0
- RUBY 2.X Universal RCE Deserialization Gadget Chain
- CVE-2018-8421 Bypassing Microsoft XOML Workflows Protection
- CVE-2018-2628, CVE-2018-2893 Oracle WebLogic Server
- CVE-2018-0147 Cisco
- BlackHat USA 2018 Sam Thomas “It’s A PHP Unserialization Vulnerability Jim, But Not As We Know It”

What is Serialization?

- a.k.a. “marshaling”, “pickling”, “freezing”, ”flattening”



- Formats
  - Binary (Java Serialization, Ruby Marshal, .NET BinaryFormatter, Protobuf), Hybrid/Other (PHP Serialization, Python pickle, Binary XML/JSON) , Readable (XML, JSON, YAML)

Why and where

- Remote/Interprocess Communication (RPC/IPC)
  - Wire protocols, web services, message brokers
- Caching/Persistence
  - Databases, cache servers, file systems
- Tokens
  - HTTP cookies, HTML form parameters, API auth tokens

No magic

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00  .....ÿÿÿÿ.......
00000010  00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72  ......;Misc, Ver
00000020  73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000030  6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50  lture=neutral, P
00000040  75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000050  6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A  ll......Serializ
00000060  61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F  ableClass.....So
00000070  6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00  meValue.........
00000080  00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B     ..Hello World!.
```

Library name

No magic

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00  .....ÿÿÿÿ.......
00000010  00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72  ......;Misc, Ver
00000020  73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000030  6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50  lture=neutral, P
00000040  75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000050  6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A  ll......Serializ
00000060  61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F  ableClass.....So
00000070  6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00  meValue.........
00000080  00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B     ..Hello World!.
```

Type name

No magic

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00  .....ÿÿÿÿ.......
00000010  00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72  ......;Misc, Ver
00000020  73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000030  6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50  lture=neutral, P
00000040  75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000050  6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A  ll......Serializ
00000060  61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F  ableClass.....So
00000070  6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00  meValue.........
00000080  00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B     ..Hello World!.
```

Field name

No magic

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00  .....ÿÿÿÿ.......
00000010  00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72  ......;Misc, Ver
00000020  73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000030  6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50  lture=neutral, P
00000040  75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000050  6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A  ll......Serializ
00000060  61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F  ableClass.....So
00000070  6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00  meValue.........
00000080  00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B     ..Hello World!.
```

Value

What could possibly go wrong here? :)

```
public static SomeClass Deserialize(byte[] data)
{
    BinaryFormatter fmt = new BinaryFormatter();
    MemoryStream stm = new MemoryStream(data);

    return fmt.Deserialize(stm) as SomeClass;
}
```

What could possibly go wrong here? :)

```
public static SomeClass Deserialize(byte[] data)
{
    BinaryFormatter fmt = new BinaryFormatter();
    MemoryStream stm = new MemoryStream(data);

    return fmt.Deserialize(stm) as SomeClass;
}
```

You might be too late!

Just Being Malicious

```
[Serializable]
public class TempFileCollection
{
    private Hashtable files;
    // Other stuff...
    ~TempFileCollection()
    {
        foreach (string file in files.Keys)
        {
            File.Delete(file);
        }
    }
}
```

Deserialized list of files

Makes sure to delete them when object destroyed!

Property-Oriented Programming / Object Injection

- Code reuse attack (a la ROP)
- Uses “gadget” classes already in scope of application
  - Relies only on code ***available*** to application
  - Not necessarily code ***used*** by application
- Create chain of instances and method invocations

Just Being Malicious

```
Set root = new HashSet();
Set s1 = root;
Set s2 = new HashSet();
for (int i = 0; i < 100; i++) {
  Set t1 = new HashSet();
  Set t2 = new HashSet();
  t1.add("foo"); // make it not equal to t2
  s1.add(t1);
  s1.add(t2);
  s2.add(t1);
  s2.add(t2);
  s1 = t1;
  s2 = t2;
}
```

Demo

- Time for exploit and payload demo

Types of Interest in .NET 4

Library	Serializable	ISerializable	Callbacks	Finalizable
mscorlib	681	268	56	2
System	312	144	13	3
System.Data	103	66	1	2
System.Xml	33	30	0	0
Management	68	68	0	4

With System.Runtime.Serialization.ISerializationSurrogate .NET 3.0:

Attacker is no longer limited to serializable annotated types

Is JSON any better?

JSON Libraries

Name		Language	Type Name	Type Control	Vector
FastJSON		.NET	Default	Cast	Setter
Json.Net		.NET	Configuration	Expected Object Graph Inspection	Setter Deser. callbacks
FSPickler		.NET	Default	Expected Object Graph Inspection	Setter Deser. callbacks
Sweet.Jayson		.NET	Default	Cast	Setter
JavascriptSerializer		.NET	Configuration	Cast	Setter
DataContractJsonSerializer		.NET	Default	Expected Object Graph Inspection + whitelist	Setter Deser. callbacks
Jackson		Java	Configuration	Expected Object Graph Inspection	Setter
Genson		Java	Configuration	Expected Object Graph Inspection	Setter
JSON-IO		Java	Default	Cast	toString
FlexSON		Java	Default	Cast	Setter
GSON		Java	Configuration	Expected Object Graph Inspection	Setter

Json.Net

- It does not include Type discriminators unless `TypeNameHandling` setting other than `None` is used:
  - a)

```
var deser = JsonConvert.DeserializeObject<Expected>(json, new
JsonSerializerSettings
{
    TypeNameHandling = TypeNameHandling.All
});
```

  - b)

```
[JsonProperty(TypeNameHandling = TypeNameHandling.All)]
public object Body { get; set; }
```

Json.Net

- Use `SerializationBinder` to whitelist Types if `TypeNameHandling` is required
- Performs a verification of expected type
  - Weak: checks if assignable
  - Is not 100% bullet proof
  - `System.Data.EntityKeyMember` or derived may not need `TypeNameHandling` set to non `None`

JavaScriptSerializer

- System.Web.Script.Serialization.JavaScriptSerializer
- By default, it will not include type discriminator information
  - Type Resolver can be used to include this information.

```
JavaScriptSerializer sr = new JavaScriptSerializer(new SimpleTypeResolver());
string reqdInfo = apiService.authenticateRequest();
reqdDetails det = (reqdDetails)(sr.Deserialize<reqdDetails>(reqdInfo));
```

- It can be used securely as long as a type resolver is not used or the type resolver is configured to whitelist valid types.

DataContractJsonSerializer

- System.Runtime.Serialization.Json.DataContractJsonSerializer
- Performs a strict type graph inspection and whitelist creation.
- However, if the attacker can control the expected type used to configure the deserializer, they will be able to gain code execution:

```
var typename = cookie["typename"];
…
var serializer = new DataContractJsonSerializer(Type.GetType(typename));
var obj = serializer.ReadObject(ms);
```

- Can be used securely as long as the expected type cannot be controlled by users.

.NET Native Formatters

Name		Format	Additional requirements	Comments
BinaryFormatter		Binary	No	ISerializable gadgets
SoapFormatter		SOAP XML	No	ISerializable gadgets
NetDataContractSerializer		XML	No	ISerializable gadgets
JavaScriptSerializer		JSON	Insecure TypeResolver	Setters gadgets
DataContractSerializer		XML	Control of expected Type or `knownTypes` or weak `DataContractResolver`	Setters gadgets Some ISerializable gadgets
DataContractJsonSerializer		JSON	Control of expected Type or `knownTypes`	Setters gadgets Some ISerializable gadgets
XmlSerializer		XML	Control of expected Type	Quite limited; does not work with interfaces
ObjectStateFormatter		Text, Binary	No	Uses BinaryFormatter internally; TypeConverters gadgets
LosFormatter		Text, Binary	No	Uses ObjectStateFormatter internally
BinaryMessageFormatter		Binary	No	Uses BinaryFormatter internally
XmlMessageFormatter		XML	Control of expected Type	Uses XmlSerializer internally

Attacking any deserializer

- It is not specific to deserialization format since deserialized objects will need to be created and populated somehow, i.e. calling:
  - Setters
  - Constructors
- Arbitrary Code Execution Requirements:
  - 1. Attacker can control type to be instantiated upon deserialization
  - 2. Methods are called on the reconstructed objects
  - 3. Gadget space is big enough to find types we can chain to get RCE
- Gadgets can be reused in different serializers and formats

Trust Issue

- This is not a new problem
- This is not a language problem
- This is not a format problem
- This is not a gadget problem
  - More will be always found
- This is trust issue

Mitigations

1. Do not deserialize untrusted data

Mitigations

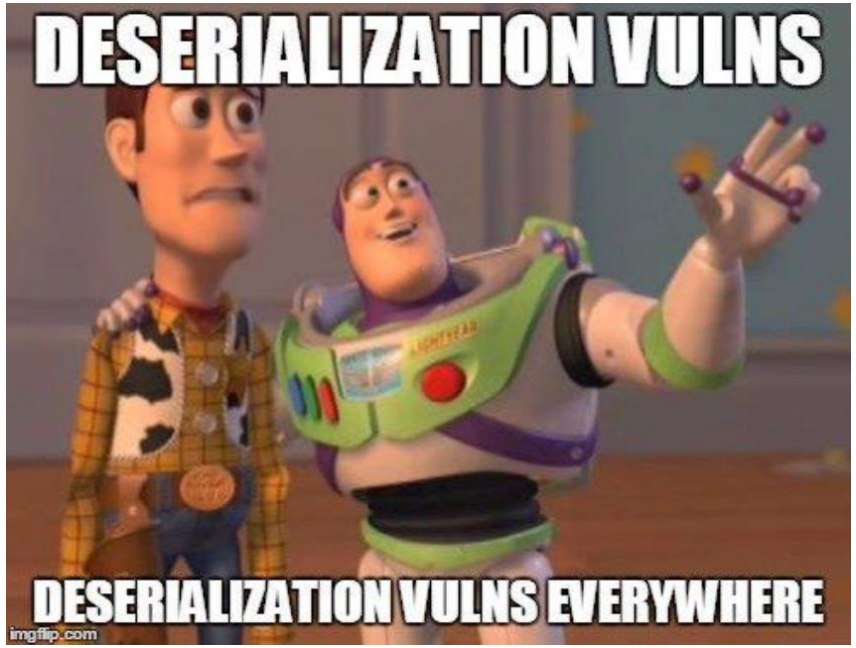
1. Do not deserialize untrusted data
2. No, seriously, don’t do it!

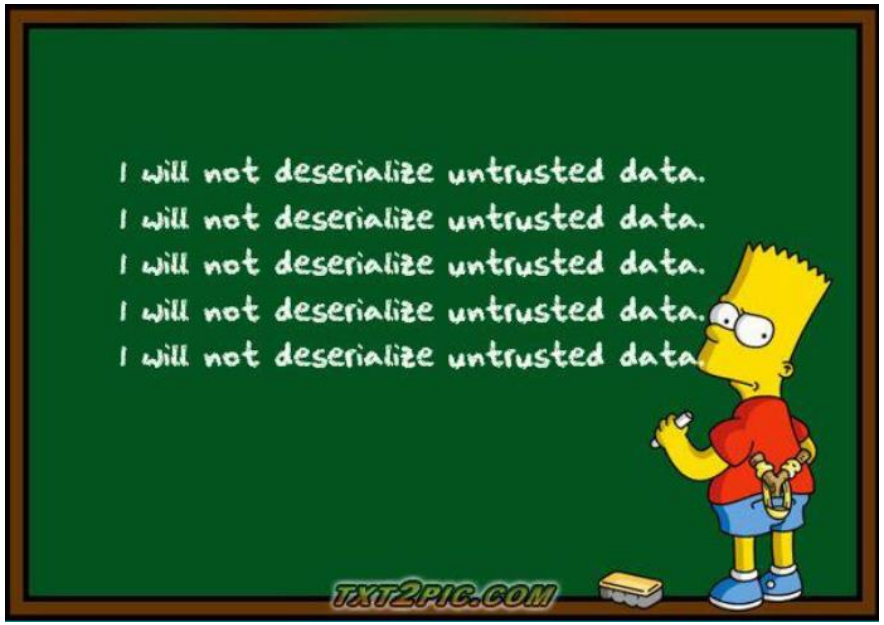
Mitigations

1. Do not deserialize untrusted data.
2. Never use user-controlled data to define the deserializer expected Type.
3. Whitelist types allowed for deserialization.
   - Can be difficult to implement.
   - Whitelist vs Blacklist
4. Use simple Data Transfer Objects (DTO) for serialization.
5. Use safe serialization library. Don’t enable unsafe configuration settings.
6. Use HMAC or ***Authenticated*** encryption to verify serialized data integrity.
   - Can’t be done on the client side.
   - Must be verified pre-deserialization!

Tools for (Pen)testers

- https://github.com/pwntester/ysoserial.net
- https://github.com/frohoff/ysoserial
- https://github.com/mbechler/marshalsec
- https://github.com/nccgroup/freddy





Thank you! Questions?

jarlob@gmail.com

https://www.linkedin.com/in/yarlob/

https://twitter.com/yarlob

References

- 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.pdf
- http://frohoff.github.io/appseccali-marshalling-pickles/
- http://frohoff.github.io/owaspsd-deserialize-my-shorts/
- https://www.slideshare.net/codewhitesec/java-deserialization-vulnerabilitesruhrseceditionv10
- https://www.youtube.com/watch?v=9Bw1urhk8zw
- https://www.owasp.org/index.php/Deserialization_Cheat_Sheet
- https://blog.scrt.ch/2016/05/12/net-serialiception/
- https://media.blackhat.com/bh-us-12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_Slides.pdf
- https://media.blackhat.com/bh-us-12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_WP.pdf
- https://googleprojectzero.blogspot.lt/2017/04/exploiting-net-managed-dcom.html
- https://bugs.chromium.org/p/project-zero/issues/detail?id=1081#c5
- https://www.blackhat.com/docs/us-16/materials/us-16-Kaiser-Pwning-Your-Java-Messaging-With-Deserialization-Vulnerabilities-wp.pdf