History of Deserialization

RCE for the modern web applications

Presentation by Philippe Arteau

GoSecure
POWERED BY COUNTERTACK

Who I Am

- Philippe Arteau
- Security Researcher at GoSecure POWERED BY COUNTERTACK
- 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 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

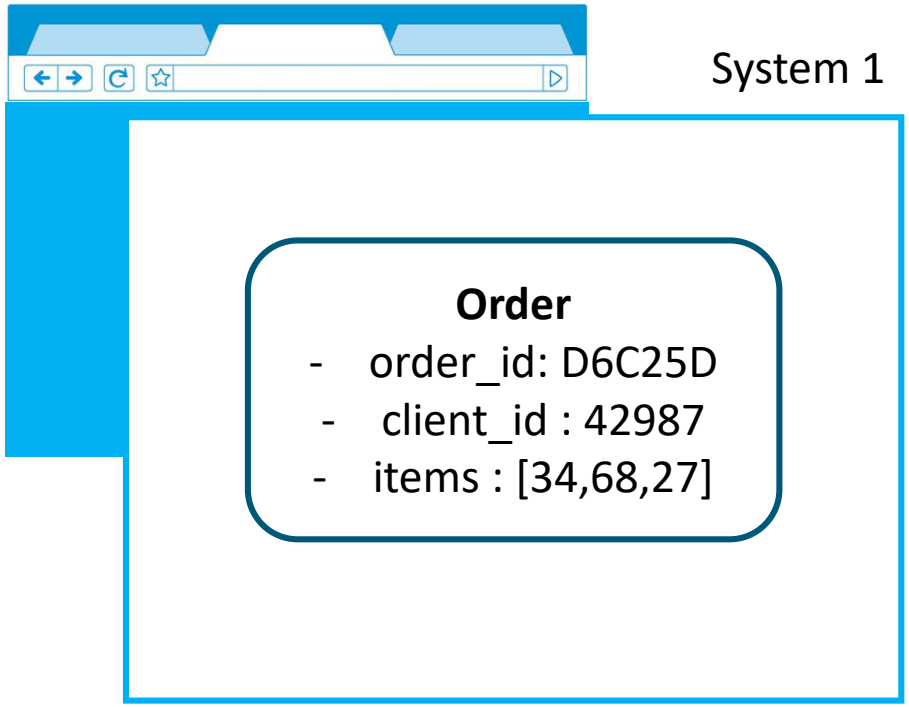
Deserialization

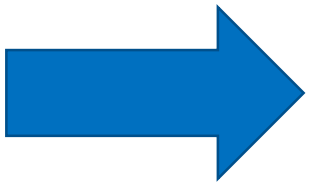
Definition

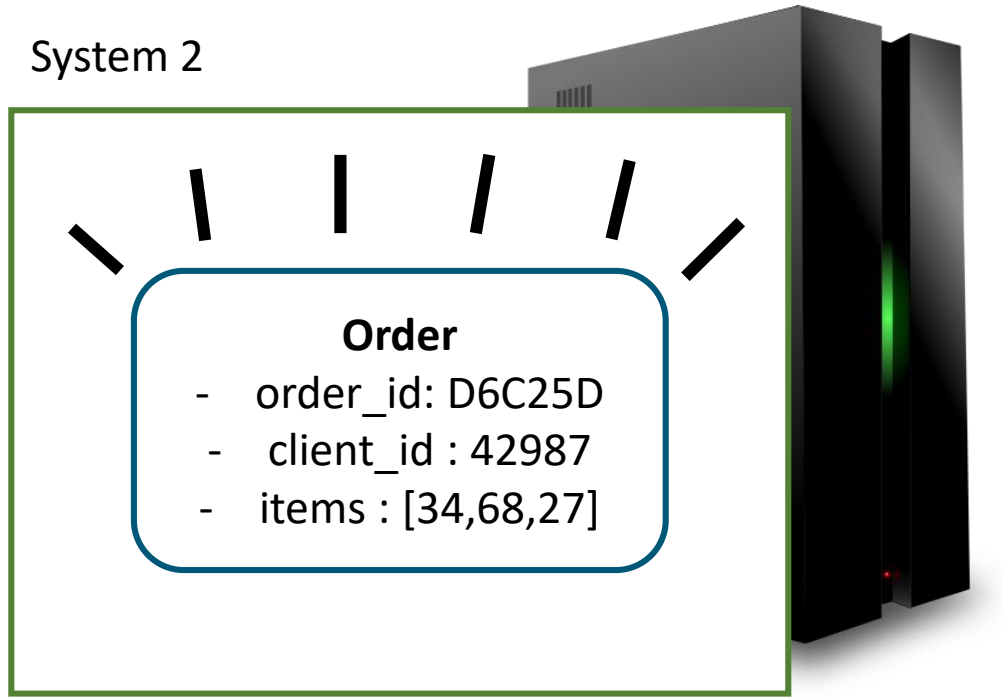
"Serialization is the process of translating data structures or object states into a format that can be stored and **reconstructed** later in the same or another computer environment."

[Ref : Wikipedia]

Deserialization Use Cases







- 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.

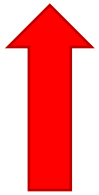
Libraries do their best to minimize side effects.

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

```
$result = unserialize($_GET['input'])
```



Unsafe deserialization

Gadget

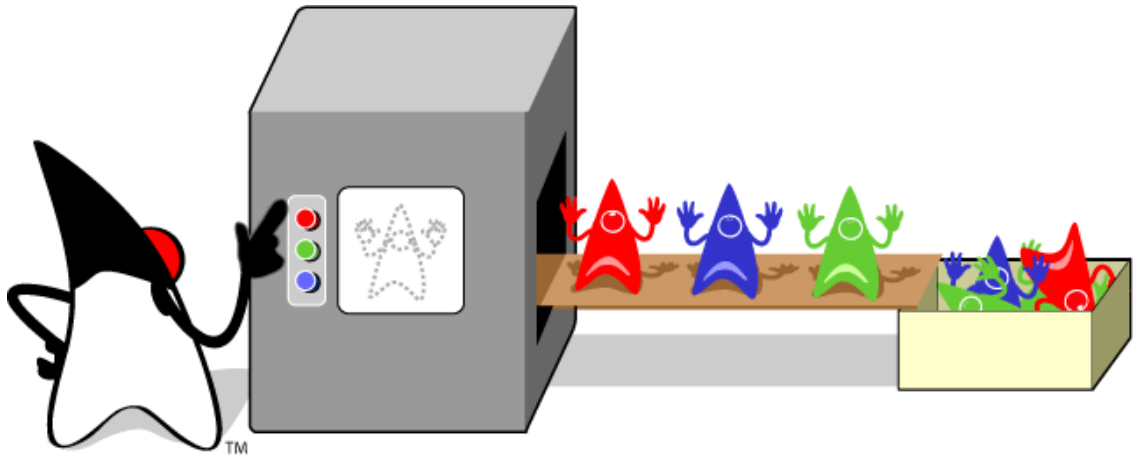
```
class sql_db {
    function __destruct() {
       $this->sql_close();
    }

    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);
    }
}
```

Specific Example: Java Native Serialization

```
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

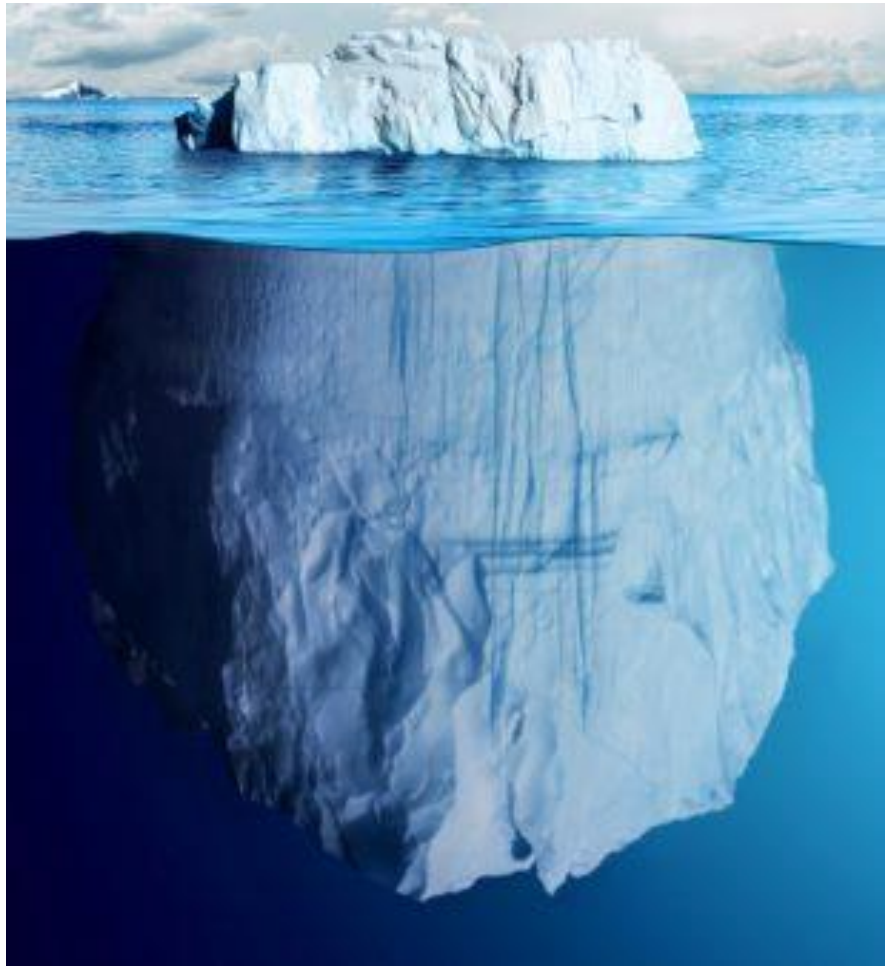
The Attack Surface

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)

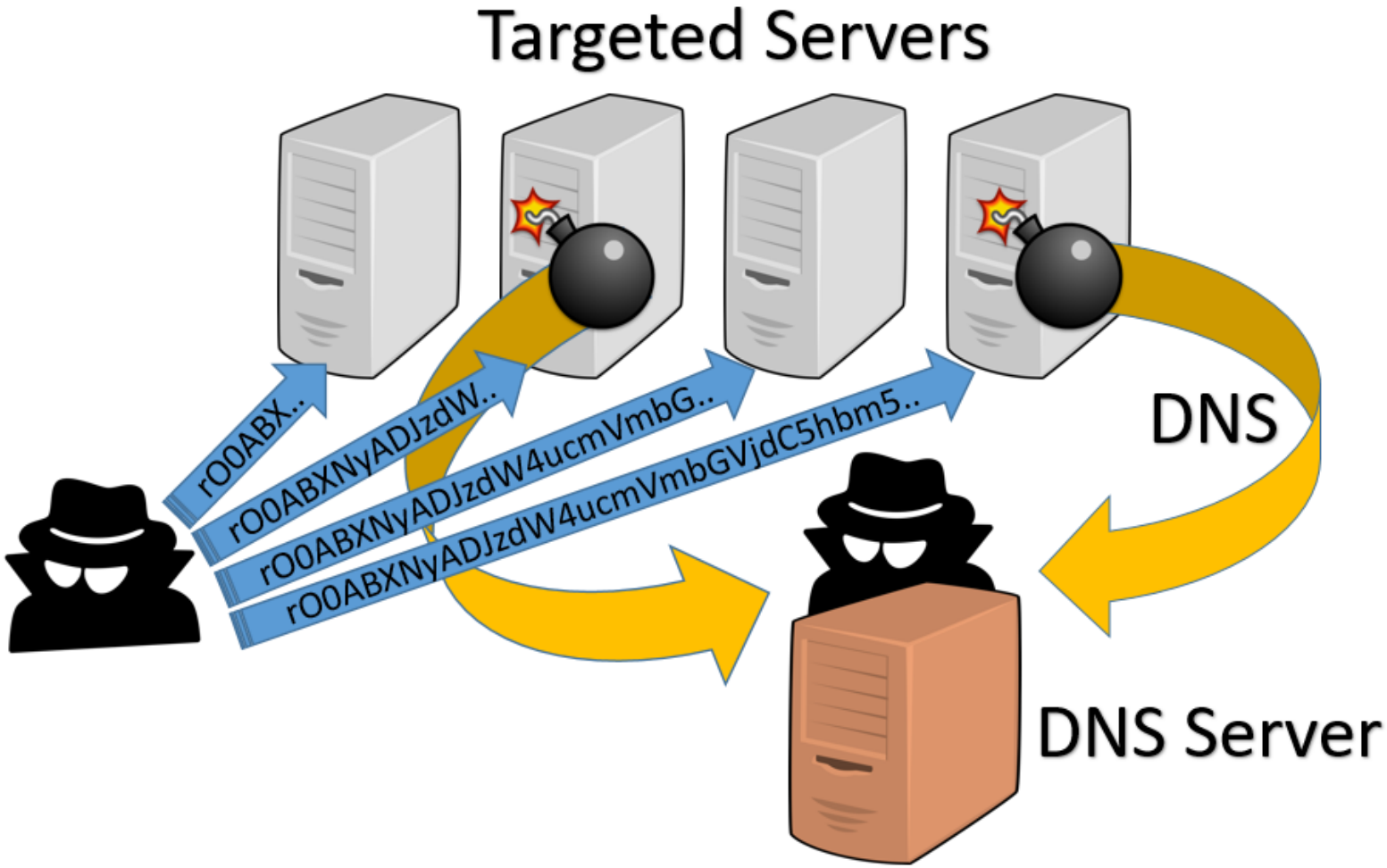
ASP.net Exploitation

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

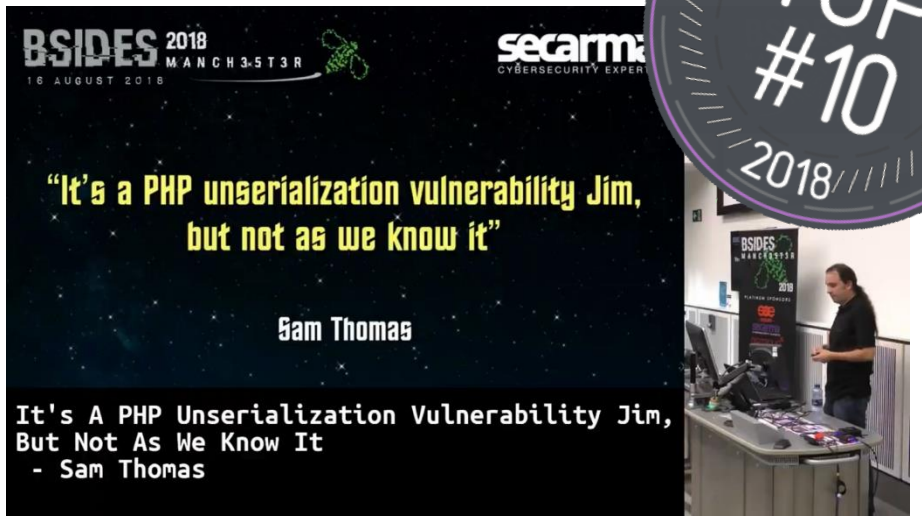
Example:

$ 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.

New PHP Exploitation Trick (2018)

- A new deserialization vector was found in PHP recently.
- It concern user input being passed to:
  - fopen()
  - copy()
  - file_exists()
  - filesize()



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

The metadata from the **PH**P **Ar**chive (PHAR) is **serialized**

https://github.com/s-n-t/presentations/blob/master/us-18-Thomas-It's-A-PHP-Unserialization-Vulnerability-Jim-But-Not-As-We-Know-It-wp.pdf

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.

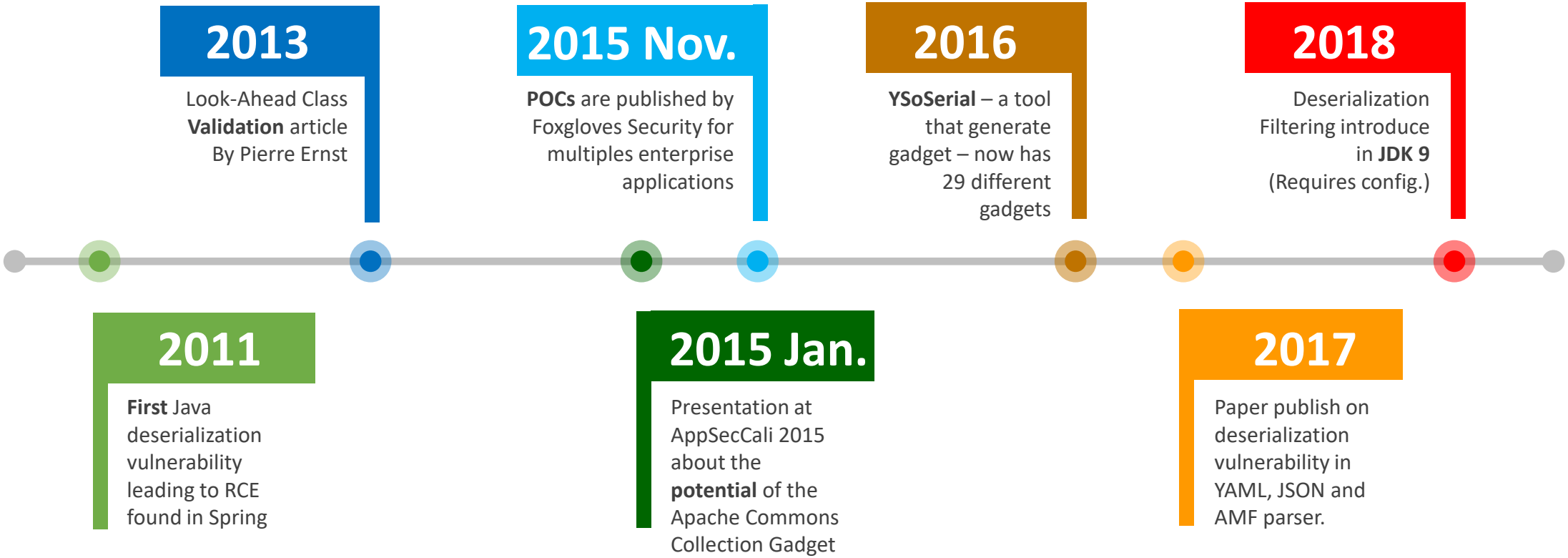
First “Gadget Based” Vulnerability

- 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.

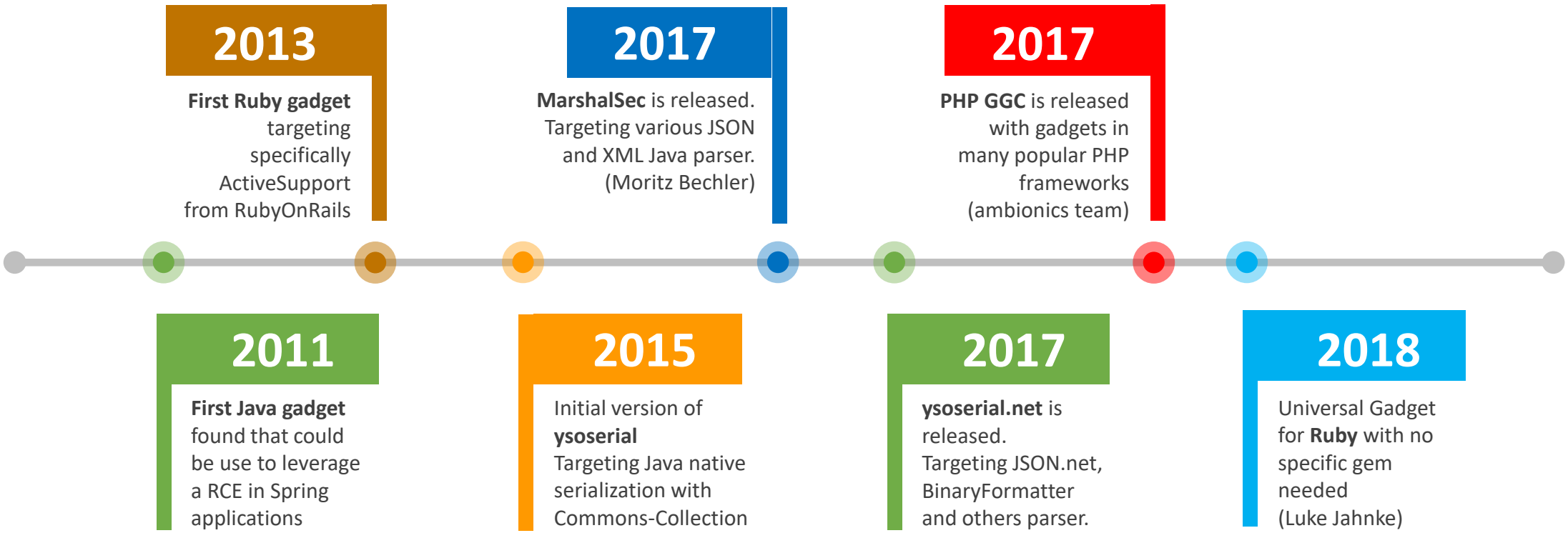
- http://www.pwntester.com/blog/2013/12/16/cve-2011-2894-deserialization-spring-rce/

Important dates in Java Deserialization History



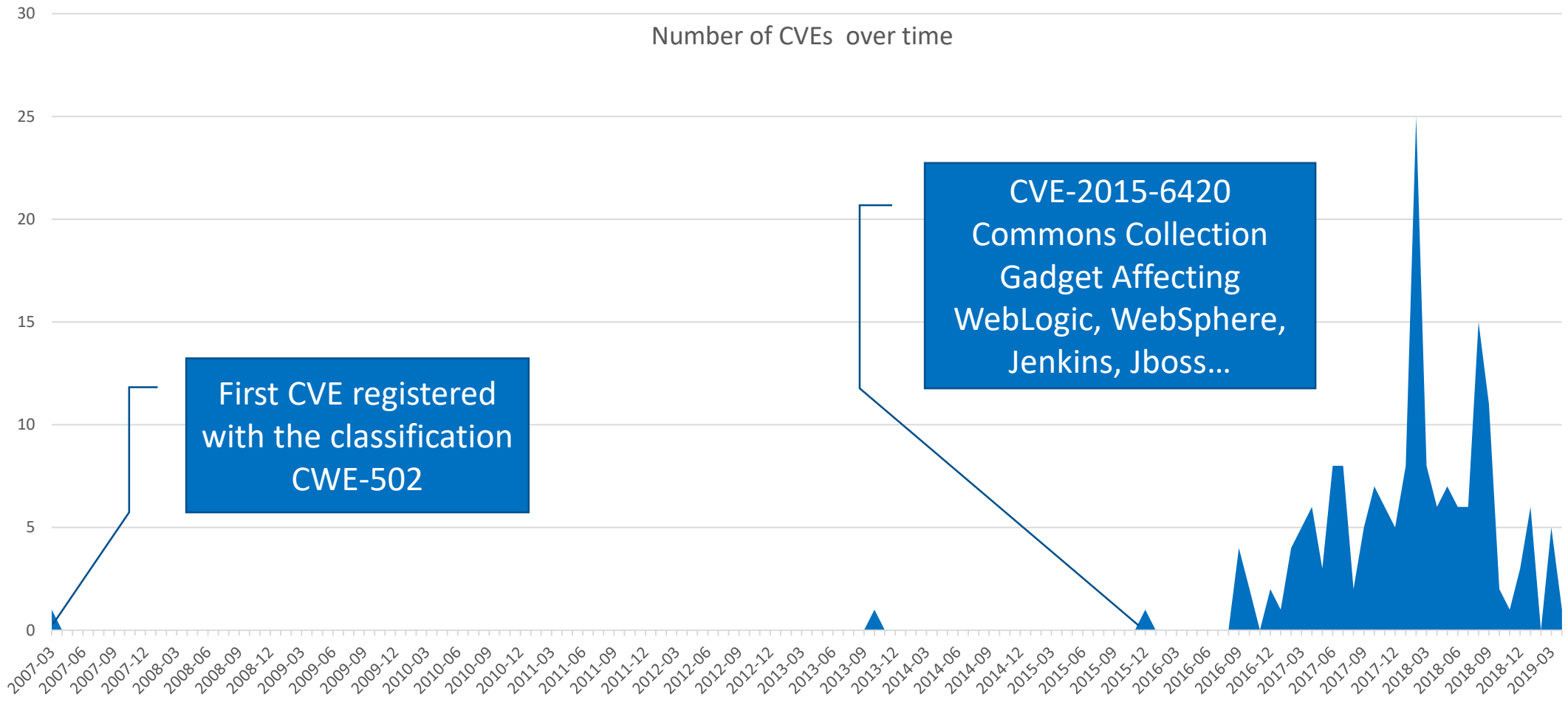
Ref: All the articles are in the references section

Gadgets timeline in Ruby, Java, .NET and PHP



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

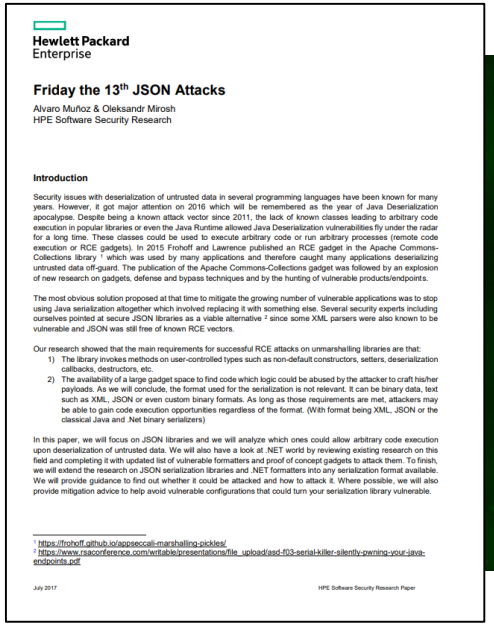
What Will Happen Next?

- 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)**



Hewlett Packard
Enterprise

Friday the 13th JSON Attacks

Alvaro Muñoz & Oleksandr Mirosh
HPE Software Security Research

Introduction

Security issues with deserialization of untrusted data in several programming languages have been known for many years. However, it got major attention on 2016 which will be remembered as the year of Java Deserialization apocalypse. Despite being a known attack vector since 2011, the lack of known classes leading to arbitrary code execution in popular libraries or even the Java Runtime allowed Java Deserialization vulnerabilities fly under the radar for a long time. These classes could be used to execute arbitrary code or run arbitrary processes (remote code execution or RCE gadgets). In 2015 Frohoff and Lawrence published an RCE gadget in the Apache Commons-Collections library [1] which was used by many applications and therefore caught many applications deserializing untrusted data off-guard. The publication of the Apache Commons-Collections gadget was followed by an explosion of new research on gadgets, defense and bypass techniques and by the hunting of vulnerable products/endpoints.

The most obvious solution proposed at that time to mitigate the growing number of vulnerable applications was to stop using Java serialization altogether which involved replacing it with something else. Several security experts including ourselves pointed at secure JSON libraries as a viable alternative [2] since some XML parsers were also known to be vulnerable and JSON was still free of known RCE vectors.

Our research showed that the main requirements for successful RCE attacks on unmarshalling libraries are that:

1) The library invokes methods on user-controlled types such as non-default constructors, setters, deserialization callbacks, destructors, etc.
2) The availability of a large gadget space to find code which logic could be abused by the attacker to craft his/her payloads. As we will conclude, the format used for the serialization is not relevant. It can be binary data, text such as XML, JSON or even custom binary formats. As long as those requirements are met, attackers may be able to gain code execution opportunities regardless of the format. (With format being XML, JSON or the classical Java and .Net binary serializers)

In this paper, we will focus on JSON libraries and we will analyze which ones could allow arbitrary code execution upon deserialization of untrusted data. We will also have a look at .NET world by reviewing existing research on this field and completing it with updated list of vulnerable formatters and proof of concept gadgets to attack them. To finish, we will extend the research on JSON serialization libraries and .NET formatters into any serialization format available. We will provide guidance to find out whether it could be attacked and how to attack it. Where possible, we will also provide mitigation advice to help avoid vulnerable configurations that could turn your serialization library vulnerable.

[1] https://frohoff.github.io/appseccali-marshalling-pickles/
[2] https://www.rsaconference.com/writable/presentations/file_upload/asd-f03-serial-killer-silently-pwning-your-java-endpoints.pdf

July 2017 HPE Software Security Research Paper

Using Safe(r) Libraries

Name	Language	Type Discriminator	Type Control	Vector
FastJSON	.NET	Default	Cast	Setter
Json.Net	.NET	Configuration	Expected Object Graph Inspection (weak)	Setter Deser. Callbacks Type Converters
FSPickler	.NET	Default	Expected Object Graph Inspection (weak)	Setter Deser. callbacks
Sweet.Jayson	.NET	Default	Cast	Setter
JavascriptSerializer	.NET	Configuration	Cast	Setter
DataContractJsonSerializer	.NET	Default	Expected Object Graph Inspection (strong)	Setter Deser. callbacks
Jackson	Java	Configuration	Expected Object Graph Inspection (weak)	Setter
Genson	Java	Configuration	Expected Object Graph Inspection (weak)	Setter
JSON-IO	Java	Default	Cast	toString

- 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

Use Blacklist or Whitelist Mechanisms

- Libraries may contains configurable whitelist and blacklist
  - Xstream (Java): allowTypeHierarchy, allowTypesByRegExp, …
  - JSON.net (C#): ContractResolver
- 3rd party libraries could be use to accommodate
  - NotSoSerial, contrast-rO0, commons-io (class ValidatingObjectInputStream)

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

[1] https://www.blackhat.com/docs/us-16/materials/us-16-Kaiser-Pwning-Your-Java-Messaging-With-Deserialization-Vulnerabilities-wp.pdf

Takeaways

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

Java 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

PHP References

- 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
- Universal Ruby Gadget https://www.elttam.com.au/blog/ruby-deserialization/

.NET References

- Ysoserial.net : Payload generator

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

- Friday The 13th JSON Attack - White Paper

https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-JSON-Attacks-wp.pdf

- New attack vector in .NET
  https://illuminopi.com/assets/files/BSidesIowa_RCEvil.net_20190420.pdf