

# Web Application Security Analysis and Implementation on Cross-Site Scripting (XSS)

Matthew Kuo  
Department of Computer Science  
Georgia State University  
Atlanta, USA  
hfkuum@gmail.com

**Abstract**—OWASP Top 10 common web security risks and Cross-Site Scripting

**Keywords**—Web Security, XSS, Cross-Site Scripting, Injection, OWASP

## I. INTRODUCTION

Because of the advancement and rapid growth of Information Technology, Cyber Security has become one of the most urgent and important issues. Among all the different research areas, Web Application is the closest concern to general public. We are all deeply susceptible to the impact from vulnerabilities in websites, interfaces and so on. From online banking to booking a flight ticket, people living in the modern society are heavily dependent on services that have the possibilities to be exploited and manipulated, which can lead to tremendous amount of loss. To raise public and corporate awareness of the importance of web application security, the Open Web Application Security Project (OWASP) [1] started a project to analyze top ten most critical web application security risks. Through the feedbacks from professional readers and the community, and constant improvement of the OWASP team, the report has now become a testable standard for Web Security.

The purpose of this paper is to introduce and explain, in a detailed way, how these threats are carried out and how to effectively defend against these threats. In order to deliver these concepts, background knowledge and foundational theories will be included and illustrated. At the last part, there will be implementations on these attacks and defenses. By reading the paper, readers will be able to establish a primitive but comprehensive understanding of Web Security.

## II. THREATS

### A. Injection & Defenses

Injection is that attackers send maliciously, carefully, and accurately coined data into an interpreter to run as command or query without proper authorization. Common injections include SQL, XPath, Buffer Overflow, LDAP, and so on.

To illustrate such idea, I will explain SQL injection attack as an example. To conduct SQL injection,

attackers submit SQL syntax or commands using form or input box [2], so that these SQL queries are passed to and executed by the databases behind the victim website. SQL injection allows attackers to obtain sensitive information from databases, to pass authentication, to escalate privilege, or to impersonate certain users that they are not.

The below command is a simple example of how attackers can make use of injecting SQL query to pass authentication. Suppose there's a website, which doesn't check SQL injection attacks, asking user to enter their username and password to authenticate their identity:

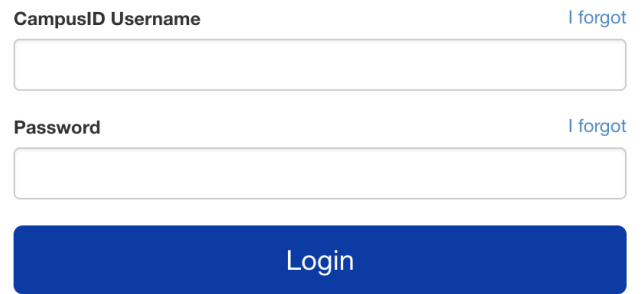


Figure 1. login page of Georgia State University PAWS system

Attackers can enter K' OR '1 = 1' in both forms. The command will always return TRUE because 1 is always equal to 1, so that attackers can pass the authentication if the website or server doesn't check whether the input data will be executed as a part of the program or not. Similar techniques can also be used to insert, steal, delete, and modify data in the victim database.

To defend against such attacks, there are different approaches which can achieve the same goals.

#### 1. Disable certain characters as input data:

Since SQL injection is to inject SQL queries or commands into the web application, we can simply disallow characters such as "'", "/", and "=" . This way, attackers will not be able to manipulate data entry to trick the web application into executing such command or statement.

#### 2. MySQL Escaping

MySQL supports two escaping modes:

ANSI\_QUOTES SQL mode and MySQL mode.

Escape Sequence	Character Represented by Sequence
\0	An ASCII NUL (x'00') character
\'	A single quote (') character
\"	A double quote (") character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character
\Z	ASCII 26 (Control+Z); see note following the table
\\	A backslash (\) character
\%	A % character; see note following the table
\_	A _ character; see note following the table

Escaping is to encode certain characters with other characters that will not be used to execute commands or query.

Figure 2. Special Character Escape Sequences [3]

### B. Broken Authentication and Defenses

Broken Authentication is that a web application doesn't correctly implement their authenticating process, which leads to the possibility for attackers to compromise usernames, passwords, keys, or session tokens, or to gain information regarding other users' identities. [4]

This often happens under the following conditions:

- Brute force: attackers keep trying pairs of username and password until they gain access to the service.
- Weak passwords: well-known or simple passwords such as "12345678", "password", or "admin".
- Weak recovery: when users try to recover their identities (username or password), the process should be hard enough so that only users themselves are able to complete.

To defend against such attacks:

- Disallow multiple authentication trials: the feature of brute force and credential stuffing is that there will be a huge amount of trials to finding the correct pair of username and password. Therefore, one of the common implementations is to limit the number of how many times users can try their password.
- Disallow weak password: the web application should check whether their users are using weak password, if so, request for a password that is strong enough.
- Session timer: kill a session when it's not been exchanging data for a certain period of time.

### C. Sensitive Data Exposure and Defenses

Sensitive data, such as health records, personal data, and credit cards, can be exposed when a web application doesn't properly protect the confidentiality of their data.

This can happen because:

- Data are not encrypted in/throughout the transmission process.
- Keys are stolen because of man-in-the-middle attacks or poor key generation and management.
- Poorly designed privacy policy from web applications: sometimes these sensitive data

may be accidentally displayed by web applications to the public or certain group of people.

To defend against such threats:

- Properly encrypt sensitive data with up-to-date and strong encryption algorithms
- Strictly impose access control
- Use secure transport protocols to guarantee confidentiality during transmission
- Using salted hashing functions to store keys

### D. XML External Entities (XXE) and Defenses

XML External Entities (XXE) happens when XML processors don't check access control, so that attackers can use maliciously designed XML files to disclose internal files. Attackers can potentially extract data, run requests from the internal service, initiate DOS attack and so on.

This kind of attack can happen when:

- XML processors don't check if the uploaded XML files are from trusted parties.
- XML processors don't check if the content of uploaded XML files contain malicious behaviors/statements.

To defend against this kind of attacks:

- Use less complicated data formats.
- Patch or modify XML processor and libraries being used.
- Use whitelist to make sure that XML files are uploaded from trusted parties.
- Manually review or automated scan codes to detect malicious behaviors.

### E. Broken Access Control and Defenses

Broken Access Control Attacks happen when a web application doesn't correctly enforce users' access privilege. Through exploiting this vulnerability, attackers may be able to access sensitive data that should be kept completely private, methods that should not be accessible to other users. What's worse, attackers may even be able to modify content of data or escalate privilege without being detected.

There are few potential causes of such attacks:

- Attackers use certain techniques to bypass access control checks.
- A web application allows administrative privilege to be given to normal users.
- Escalation of privilege is not carefully examined.

- d. Relying too much on single or few files/data to conduct primary behaviors/modifications.
- e. A web application doesn't carefully examine incoming/outgoing requests.

To defend against these attacks:

- a. Use whitelist to make sure only trusted parties can do certain actions/behaviors.
- b. Keep track of privilege records of all users, when modified in a suspicious way, report.
- c. Keep file tree absolutely private.
- d. Enforce minimal access control, which means access is given only when it's really needed.

#### F. Security Misconfiguration and Defenses

Security Misconfiguration is that the web application itself has some vulnerabilities known to attackers. These vulnerabilities can impact to system at different levels depending on which vulnerability is being exploited. Because of the growing need for cloud services, this kind of threats is more and more common.

This kind of threat can happen when:

- a. The system gave permissions on cloud services to users more than needed, which means it doesn't go by the concept of "Principle of Least Privilege".
- b. Some features that are not essential but can compromise the system are enabled. For example, unused ports, super users, commands, and so on.
- c. Sensitive information may be leaked out because of misconfigured error messages.
- d. The web application relies on other services that are vulnerable.

To defend against such attacks:

- a. During each system upgrade, a detailed security testing should be conducted.
- b. Only allow needed features, the more unnecessary features are enabled, the more potential threats there will be.
- c. Follow the principle of least privilege.

#### G. Cross-Site Scripting (XSS) and Defenses

Cross-Site Scripting (XSS) is that attackers inject malicious codes into a web application so that the codes can be run and executed by victims' browser. This usually happens when there are forms for users to enter data on a web application, but input is not checked whether it contains executable statements. Therefore, the malicious user-supplied data become an executable part of the whole application that can

potentially steal information from victims or change the whole website.

There are three types of Cross-Site Scripting:

- a. Reflected XSS: the web application uses certain services that contain malicious user input that can execute HTML or JavaScript in the victims' browser.
- b. DOM XSS: attackers modify certain the environments of the client-side script, which results in that the same payload is exchanged but different behaviors are being done.
- c. Stored XSS: user-supplied input is stored as a part of the web application. This kind of XSS is specifically dangerous because malicious codes will continually be in effect till they are noticed and deleted.

To defend against XSS:

- a. Encode user-supplied input. When there are certain characters being entered such as "'", "/", "<", ">", or ";", replace them with other characters that will not be executable.
- b. Block untrusted JavaScript requests to sites with different domain from the web application. This can be achieved by whitelisting.

#### H. Insecure Deserialization and Defenses

Insecure Deserialization is that maliciously designed data are used to interrupt or disturb application logic, deny service, or execute arbitrary code. [5]

This happens when:

- a. Attackers modify certain meaningful objects, data, or classes so that the system will take these commands and statements as usual. Therefore, remote code execution is achieved.
- b. Attackers modify application logic so that attackers can bypass or tamper the correct application logic and use this vulnerability to gain access or perform DOS attacks.

To defend against the threat:

- a. Disallow objects from untrusted sources.
- b. Only accept primitive data types.
- c. Use digital signature to validate the authenticity of source.
- d. Use sandbox-like system to run the code to examine the behaviors being done

- e. Filter network connection so that even malicious codes are executed, they can be controlled and contained.

#### *I. Using Components with Known Vulnerabilities and Defenses*

Nowadays, most web applications rely heavily on services provided by different parties. In order to achieve these services' functionality, a lot of them have to run with high privilege. Therefore, if there are known vulnerabilities of these services, the web applications using the services are also susceptible to the same threats.

This happens when:

- a. Developers don't keep track of what services they use and who provide them, so that it's impossible to patch the components accordingly.
- b. Components are not properly maintained
- c. Service provider can't fix these reported issues in a timely fashion
- d. Upgrade services from an untrusted or malicious source (spoofing attack)

To defend against such attacks:

- a. Disable features that are not really needed, files, and records.
- b. Keep track of versions of all the services that are used, so that we can see where to patch.
- c. Only get services (either first time or upgrade) from trusted source.
- d. Actively search and scan for potential threats in services provided by other parties.

#### *J. Insufficient Logging & Monitoring and Defenses*

When suspicious or even malicious behaviors happen, the system should automatically record, in detail, such events because these records provide valuable information for analyst to find out where the problems or threats are. However, a lot of event recorders now don't do so in a timely fashion. What's worse, sometimes what they record is not complete enough to reconstruct what really happened and how attacks are carried out. As a result, attackers will have enough time and opportunities to finish their intrusion, attacks, or probing.

This happens when:

- a. System doesn't record important events, such as failed logins, file upload, port scanning, and so on.

- b. Services provided by other parties are not monitored.
- c. Behaviors incurred by executions or commands are not recorded.

To defend against such attacks:

- a. Make sure that the system records important events.
- b. Make sure that the records are easily accessible and understandable.
- c. Build an internal monitor to capture all behaviors incurred by executions or commands.

### III. CROSS-SITE SCRIPTING RELATED RESEARCH

Among all the topics, I decide to pick Cross-Site Scripting (XSS) as the topic to go further into.

#### *A. Attacks on Web Application Caused by Cross Site Scripting [7]*

Cross-Site Scripting has always been one of the most dangerous and urgent issues/vulnerabilities in web applications. This kind of attack is especially hard to detect, because malicious codes can have a normal look that runs like normal features.

To get started, we first need to know the two types of web application.

##### *1. Static web applications*

This kind of web applications don't involve web server to handle data, programs, and so on. They just present a web application as it's already established. Therefore, every user will see the same web page.

##### *2. Dynamic web applications*

This kind of web applications use application server processing server-side scripts to construct the web pages. Therefore, there will be programs running on server or client side to produce different data and web pages for different users or scenarios.

As a result, if attackers make use of the fact that they can execute commands through such web application, potential vulnerabilities, such as XSS, SQL Injection, XML Injection, may be exploited.

Cross-Site Scripting often happens when a web application allows user to enter information into the database. Since users have the liberty to put whatever into the web server, attackers will make use of this opportunity to inject malicious codes. Without proper defenses or filtering, these malicious codes can become a part of the web application and infect the web application until they are found and deleted.

Which is to say, this type of attack can last for a long time, impact a lot of users, if the web application is not well maintained.

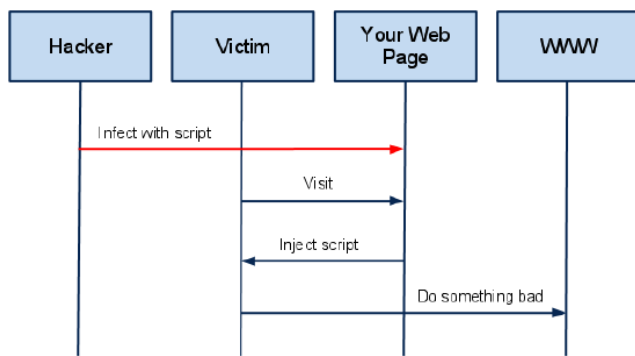


Figure 3. XSS Illustration [6]

As we can see from this figure. Hacker will inject malicious script into the web application, as it becomes a part of the web page, victim's browser will execute the script when visiting the infected web page. As a result, cross domain requests or connections that may be really harmful to the victim will be initiated.

To identify such threats, there are two major approaches to achieve the goal:

## 1. STATIC ANALYSIS

Static analysis is to use flow sensitive analysis and context sensitive data flow to detect possible logic flaws that can lead to Cross-Site Scripting attacks. Additionally, as more and more vulnerabilities are discovered, analysts can also put different weights on different behaviors, so that the report will become more and more accurate.

Currently, there are a lot of open source projects that can detect Cross-Site Scripting with static analysis.

- Google CodeSearchDiggity
- Progpilot
- .NET Security Guard
- ...

## 2. DYNAMIC ANALYSIS

As for dynamic analysis, it is mainly about memory security system. At run time, we monitor malicious behaviors and their subsequent polluted memory usage. For example, if there is a request initiated toward outside domain, it's potentially a Cross-Site Scripting attack or Cross-Site Request Forgery. Such request should be examined before sent out.

### B. Server side defense methods against Cross-Site Scripting [7]

There are multiple ways for server side to implement defense mechanism against XSS:

#### a. Never Insert Untrusted Data Except in Allowed Locations

Since the mechanism of Cross-Site Scripting attack is to inject malicious codes into the web application and make them become a part of the executable program, web application provider should never put user-supplied input into the web application before carefully examining or filtering such input.

#### b. HTML Escape Before Inserting Untrusted Data into HTML

Since Cross-Site Scripting attack relies on injecting codes that interpreter/processor can understand and execute, the web application can try to turn these potentially malicious codes into codes that interpreter/processor cannot understand.

```

& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2F;
  
```

Figure 4. Escape characters

#### c. JavaScript Escape Before Inserting Untrusted Data into JavaScript Data Values

Sometimes, web application developers may want user-supplied input to become a part of their JavaScript Codes. For example, the web application may ask users to enter their age, which will be a part of the JavaScript if statement to decide whether the user is at legal age or not. As a result, if the web application doesn't escape user-supplied input, attackers may use the carefully designed malicious codes to conduct certain harmful behaviors.

To achieve such goal, there are a lot of open source projects that can prevent XSS by serializing string literals so that when put in HTML elements, they will not be viewed as executable programs.

Take the serialization project from Yahoo for example [8], string literals will go through pre-defined rules and become safe to be put in HTML elements.

```
var serialize = require('serialize-javascript');

serialize({
  str : 'string',
  num : 0,
  obj : {foo: 'foo'},
  arr : [1, 2, 3],
  bool : true,
  nil : null,
  undef: undefined,
  date: new Date("Thu, 28 Apr 2016 22:02:17 GMT"),

  fn: function echo(arg) { return arg; },
  re: /([^\s]+)/g
});
```

Figure 5. Example input

Example output:

```
'{"str":"string","num":0,"obj":{"foo":"foo"},"arr":
[1,2,3],"bool":true,"nil":null,"date":new
Date("2016-04-28T22:02:17.156Z"),"fn":function
echo(arg) { return arg; }, "re":"/([^\s]+)/g"}
```

Example input:

```
serialize({
  haxorXSS: '</script>'
});
```

Figure 6. Serialization of certain string

Example output:

```
'{"haxorXSS":"\\u003C\\u002Fscript\\u003E"}
```

Figure 7. Output after serialization

Keep track of versions of all the services that are used, so that we can see where to patch.

- e. Only get services (either first time or upgrade) from trusted source.
- f. Actively search and scan for potential threats in services provided by other parties.

### C. Client side defense methods against Cross-Site Scripting

In this section, ways to prevent Cross-Site Scripting attacks from client side will be introduced. Because Google Chrome will block cross-site JavaScript requests in default, I will only mention extensions on Firefox.

#### 1. NoScript

This is a Firefox extension that JavaScript, Java, Flash and other plugins will be executed only when they are from trusted sources (whitelist). It implements pre-emptive script blocking approach, which mean it will stop those suspicious requests even before they leave the browser. Users are allowed to add trusted sources to their whitelist. Despite at first it will not be user-friendly since most requests are blocked, the

accuracy will grow after users' constant feedback on whitelist.

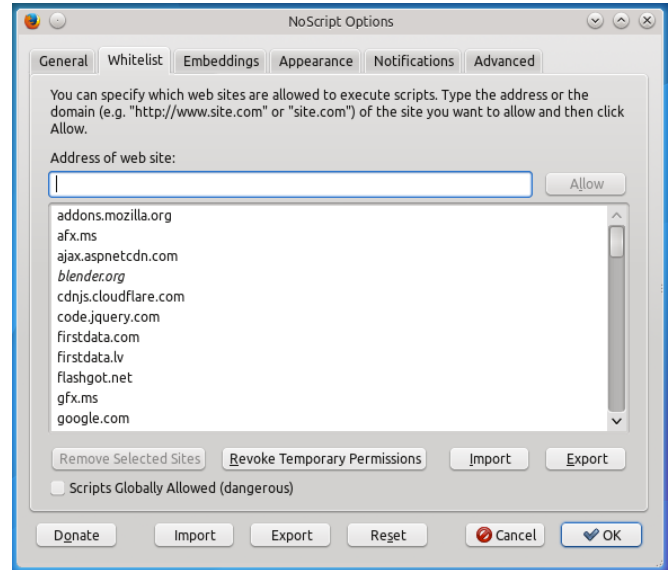


Figure 8. Whitelist for NoScript

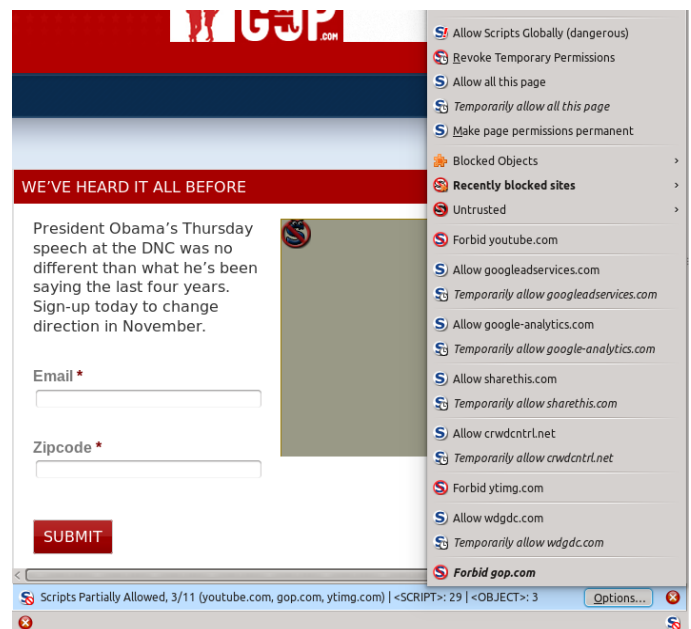


Figure 9. Interface showing current status

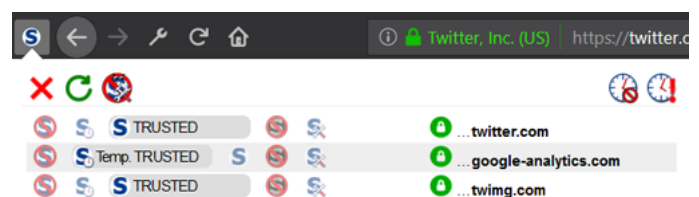


Figure 10. Trust or block option menu

#### 2. RequestPolicy Continued

RequestPolicy Continued is an open source Firefox add-on that can protect Firefox users from being



attacked by Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). The hardest part about recognizing Cross-Site Scripting attacks and Cross-Site Request Forgery attacks is that computer has no way to tell if the cross-site request is really initiated by user. Therefore, the concept behind this add-on is to used whitelisting (by default) or blacklisting to filter what are the requests that can really be initiated and sent out. For whitelisting, cross-site requests will only be sent out if the destination domain is in the whitelist. For blacklisting, cross-site requests will not be sent out if the destination domain is in the blacklist. As we all know, when it comes to whitelist and blacklist, the hardest problem is always about how to accurately come to a whitelist/blacklist that can really do the job. As a result, this Firefox add-on allows you to customize your own whitelist/blacklist.

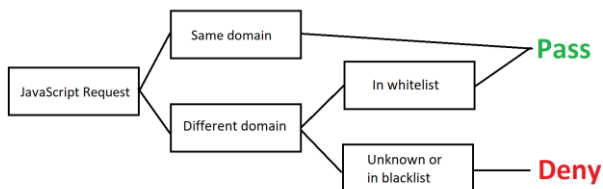


Figure 11. flow chart of how RequestPolicy Continued works

Your Policy

Create Rule

Policy ☒ Allow ☐ Block ☐ [Learn more about rules.](#)

Origin

scheme host port

Destination

scheme host port

Temporary ☐

Add rule

Figure 12. Interface that allows users to add domains to whitelist

Active Rules

Filter Rules Search

Policy	Origin	Destination	Rule Set
Allow	moz-extension://b438c401-e7cb-4e34-8480-fc923b364997	https://*	allow_extensions
Allow	moz-extension://8cbf738f-36a0-4219-8dda-d1e50eb26194	http://127.0.0.1:23119	allow_extensions
Allow	*facebook.com	*akamaihd.net	allow_functionality
Allow	*instagram.com	*d36tkk24g8jdx.cloudfront.net	allow_functionality
Allow	*instagram.com	*instagram-static.s3.amazonaws.com	allow_functionality
Allow	*microsoftstore.com	*digitalrivercontent.net	allow_functionality
Allow	*stackoverflow.com	*imgur.com	allow_functionality
Allow	*stackoverflow.com	*ajax.googleapis.com	allow_functionality
Allow	*stackexchange.com	*imgur.com	allow_functionality
Allow	*stackexchange.com	*ajax.googleapis.com	allow_functionality
Allow	*superuser.com	*imgur.com	allow_functionality
Allow	*superuser.com	*ajax.googleapis.com	allow_functionality
Allow	*serverfault.com	*imgur.com	allow_functionality
Allow	*serverfault.com	*ajax.googleapis.com	allow_functionality
Allow	*askubuntu.com	*imgur.com	allow_functionality
Allow	*askubuntu.com	*ajax.googleapis.com	allow_functionality
Allow	*dropbox.com	*dt8k5553cwe8.cloudfront.net	allow_functionality
Allow	*bitbucket.org	*d3oanc4g5k2d5q.cloudfront.net	allow_functionality
Allow	*addons.videolan.org	*vlc-addons.org	allow_functionality
Allow	*graphhopper.com	*lyrik.org	allow_functionality
Allow	*graphhopper.com	*openstreetmap.org	allow_functionality

Figure 13. Current active whitelist

Policy ☒ Allow ☐ Block ☐ [Learn more about rules.](#)

Origin

scheme host port

Destination

scheme host port

Temporary ☐

Add rule

Figure 14. Interface that allows users to add domains to blacklist

Block	*atmetric.com	deny_trackers
Block	*pagetair.com	deny_trackers
Block	*pagetair.net	deny_trackers
Block	*nexc.com	deny_trackers
Block	*4dsply.com	deny_trackers
Block	*adsfactor.net	deny_trackers
Block	*adglobal.com	deny_trackers
Block	*d5vial8fue4z.cloudfront.net	deny_trackers
Block	*pixieinteractivemedia.com	deny_trackers
Block	*regiohelden.de	deny_trackers
Block	*gorooost.com	deny_trackers
Block	*viglink.com	deny_trackers
Block	*n72adserv.com	deny_trackers
Block	*cruz.net	deny_trackers
Block	*sitetag.us	deny_trackers
Block	*giyou.com	deny_trackers
Block	*egoad.com	deny_trackers
Block	*popads.net	deny_trackers
Block	*778669.com	deny_trackers
Block	*linezing.com	deny_trackers
Block	*d31qbtv1cthecs.cloudfront.net	deny_trackers
Block	*besocialmedia2.com	deny_trackers
Block	*integral-marketing.com	deny_trackers
Block	*wwwpromoter.com	deny_trackers
Block	*ecimpsa.com	deny_trackers
Block	*directrev.com	deny_trackers
Block	*adnotch.com	deny_trackers
Block	*ad-center.com	deny_trackers
Block	*mdn2015x1.com	deny_trackers
Block	*mdn2015x2.com	deny_trackers
Block	*threads.me	deny_trackers
Block	*stamplive.com	deny_trackers
Block	*clickrev.com	deny_trackers
Block	*srnpub.com	deny_trackers
Block	*mtagmonetizationb.com	deny_trackers
Block	*duggads.com	deny_trackers
Block	*a2pub.com	deny_trackers

Figure 15. Current active blacklist

With RequestPolicy Contined installed, when clicking on a URL to a website for the first time, users will have three options. The first option is to temporarily **allow** the request for only this time. The second option is to temporarily **deny** the request. The third option is to add rule, which has six more options. They are:

1. Allow all future requests from current source domain to destination domain.
2. Temporarily allow requests from current source domain to destination domain.
3. Allow all future requests from current source domain.
4. Temporarily allow requests from current source domain.
5. Allow all future requests to destination domain.
6. Temporarily allow requests to destination domain.

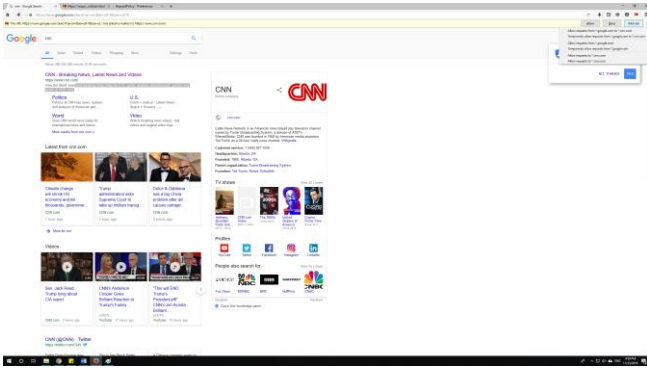


Figure 16. Options for clicking on a denied-by-default website

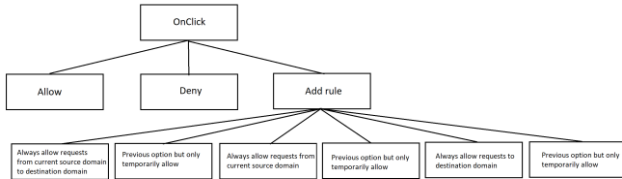


Figure 17. Options for further actions

When viewing a website, the icon on top right corner will tell users what the destination domains being denied by default are (question mark), what the destination domains being in blacklist are (red no entry sign), and what the destination domains being allowed by whitelist are (green check).

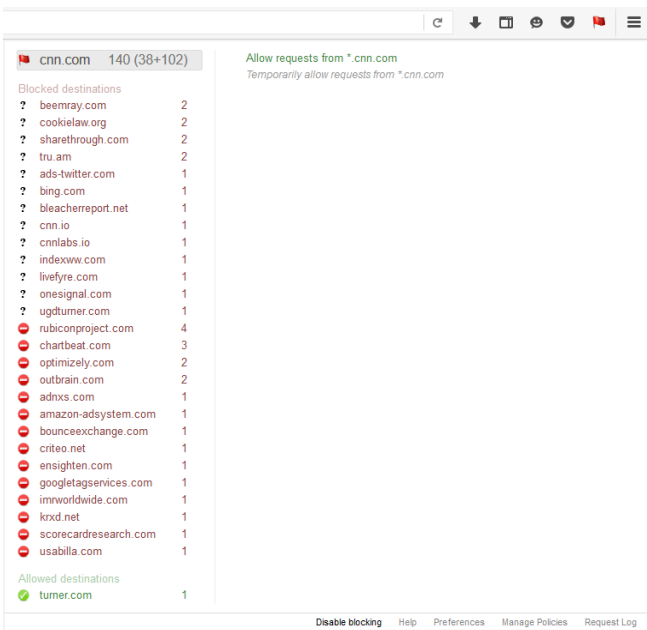


Figure 18. User interface that shows allows and denials situation.

#### D. Implementation on Cross-Site Scripting (XSS) Attacks

In order to demonstrate how a vulnerable victim website might be targeted and attacked, I built a website where visitors can enter and leave their name.

## Leave your name

Firstname:

Lastname:

Name: John Doe

Name: John Doe

Name: John Doe

Name: matt kuo

Name: haha test

Figure 19. Victim website

This kind of website is vulnerable to XSS if user-supplied data is not properly processed as mentioned earlier.

To test if the web application implements certain measures against script injection, we can simple input some test scripts. I will just enter `<script>alert(1)</script>` to check if the alert box will really be injected or not (Figure 20). However, theoretically as an attacker, we should stay as low-profile as possible, which mean using `alert()` is not a wise choice since it's too eye-catching.

## Leave your name

Firstname:

Lastname:

Name: John Doe

Name: John Doe

Name: John Doe

Name: matt kuo

Name: haha test

Figure 20. test script





Figure 21. popup alert

As we can see in figure 21, whenever users go on this website, an alert created by the injected JavaScript will pop up.

```
<html>
<body>

<h1>Leave your name</h1>
<form action="submit.php" method="post">
  Firstname: <input type="text" name="fname" /><br><br>
  Lastname: <input type="text" name="lname" /><br><br>

  <input type="submit" />
</form>

Name: John Doe<br>Name: John Doe<br>Name: John Doe<br>Name: matt kuo<br>Name: baha test<br>Name: <script>alert(1)</script> xss<br>
</body>
</html>
```

Figure 22. Source code for injected website

Although users enter data as plaintext, if input is maliciously designed, it can actually become executable program and executed by the web application. In figure 22, the `<script>alert(1)</script>` is now a part of the program of this web application.

There are many Cross-Site Scripting penetrating testing tools now, and **BeEF** is one of the most powerful among them.



Figure 23. BeEF

BeEF stands for The Browser Exploitation Framework. It can be run on Mac OSX 10.5.0 or higher, or modern Linux.

To conduct penetration test with BeEF, there will be a hook JavaScript file that I will inject into a web application.

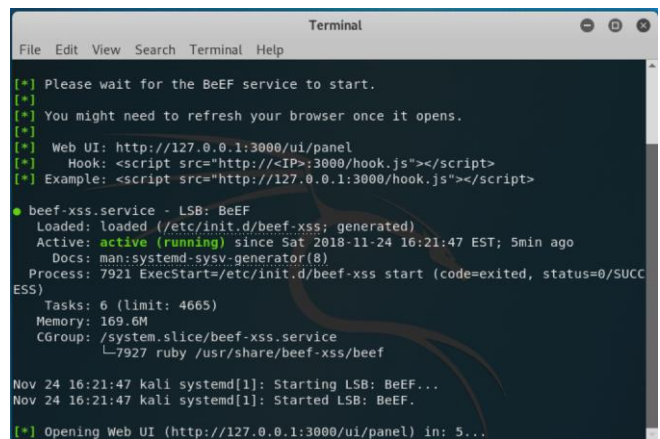


Figure 24. BeEF starting process

In this case, the script is put at:  
<http://192.168.100.130:3000/hook.js>

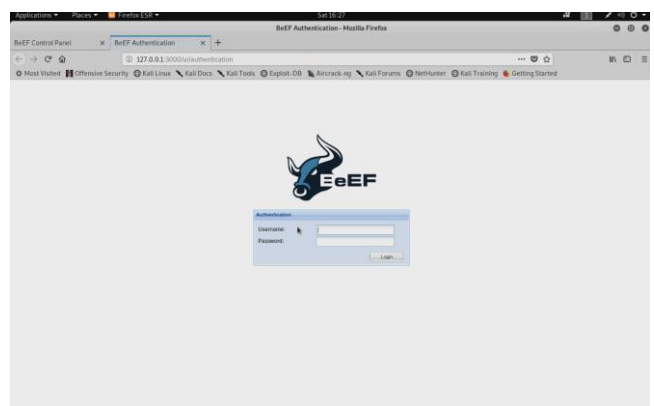


Figure 25. login page for BeEF

After loading the program, we'll reach the login page of BeEF console. The default username and password are: beef.

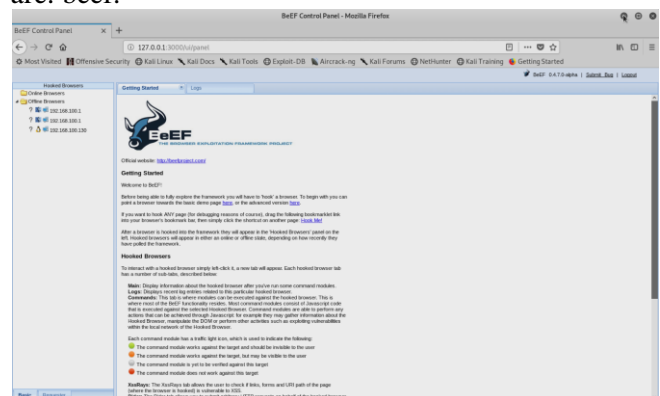


Figure 26. BeEF console

Then we can start injecting malicious script.

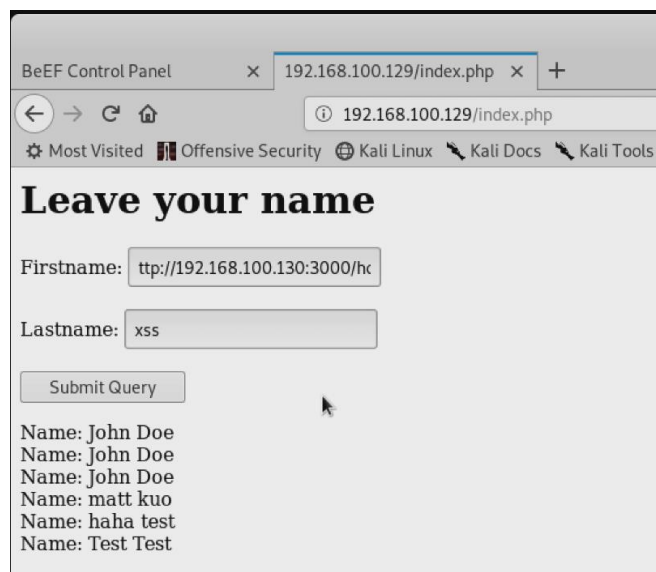


Figure 27. XSS attacks code injection

After injecting codes, we can see that it's successful by viewing the page source.

```
<html> event
<head></head>
<body> event
  <h1>Leave your name</h1>
  <form action="submit.php" method="post">
    Name: John Doe
  <br>
    Name: John Doe
  <br>
    Name: John Doe
  <br>
    Name: matt kuo
  <br>
    Name: haha test
  <br>
    Name: Test Test
  <br>
    Name:
  <script src="http://192.168.100.130:3000/hook.js"></script>
  xss
</body>
</html>
```

Figure 28. Page source with injected codes

Now, whenever there are visitors currently viewing this victim website, I will be able to see them and their information, such as Browser, Web Sockets, ActiveX.

Control Panel		Log	Current Browser					
Details		Log	Comments	Role	Access	Type	Network	WebRTC
Category: Browser (8 items)								
Browser Version: UNKNOWN								Information
Browser UA String: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4; rv:42.0) Gecko/20100101 Firefox/42.0								Information
Browser Language: en-US								Information
Browser Platform: MacIntel								Information
Browser Plugins: []								Information
Window Size: Width: 1440, height: 761								Information
Category: Browser Components (12 items)								
Flash: No								Information
VBScript: No								Information
PhoneGap: No								Information
Google Gears: No								Information
Web Sockets: No								Information
QuickTime: No								Information
RealPlayer: No								Information
Windows Media Player: No								Information
WebRTC: No								Information
ActiveX: No								Information
Session Cookies: Yes								Information
Persistent Cookies: No								Information
Category: Hosted Page (5 items)								
Page Title: Unknown								Information
Page URL: http://192.168.100.129/index.php								Information
Page Reference: Unknown								Information
Host NameSP: 192.168.100.129								Information
Cookies: BE274COK=F8E2c9m;78V87072AQ5v9vYhM;H8vF8464kxumv8v8W843v5v9FvYv8;CvL7v9W8G								Information
Category: Head (8 items)								
Head NameSP: 192.168.100.1								Information
Date: Sun, 16 Jul 2016 10:10:55.000+0100 (GMT+01:00)								Information

Figure 29. Visitors' information

Aside from digging information that can possibly be used to exploit, BeEF also allows real malicious attacks. There are many different types of attacks, including Exploit, Network, Social Engineering, etc. I will demonstrate an attack that the victim website will be replaced with a fake Google Login phishing page. In Figure 30, when I see an online visitor, I ran the Google phishing command.

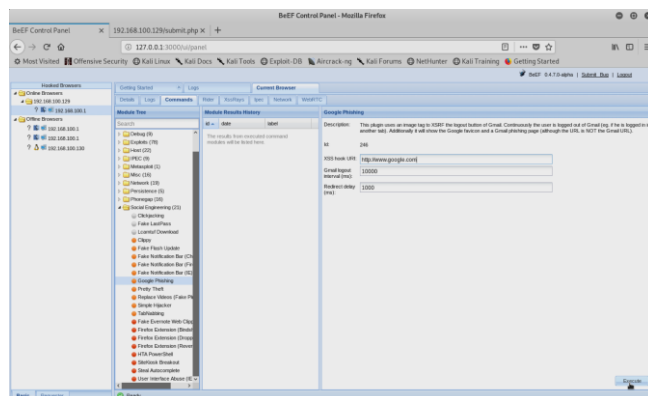


Figure 30. Attack options

As a result, the web page that the visitor was viewing had become a Google Login page.

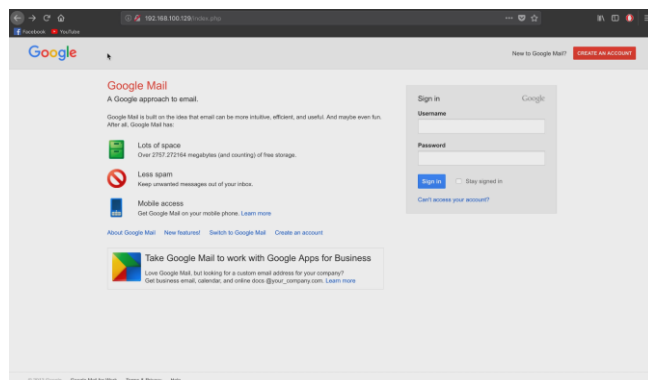


Figure 31. Fake Google Login page

I even set the redirect page after entering username and password to be the real Google page so that it's looked more convincing.

After visitor was tricked to enter their username and password for Google account, I successfully got those data.

```
1 data: result=Username: Victim Password: Victimpassword
```

### CONCLUSION

Cross-Site Scripting has always been one of the most dangerous and urgent issues/vulnerabilities in web applications. In order to prevent the growth of this kind of threats, not only server-side but also client-side should be aware of such danger and be actively defending against such attacks.

### REFERENCES

- [1] OWASP, "The Ten Most Critical Web Application Security Risks", <https://owasp.org>.
- [2] You Yu, Yuanyuan Yang, Jian Gu, and Liang Shen, "Analysis and Suggestions for the Security of Web Applications", 2011 International Conference on Computer Science and Network Technology.
- [3] Oracle Corporation MySQL Developer Zone, "String Literals", <https://dev.mysql.com/doc/refman/8.0/en/string-literals.html>.
- [4] OWASP, "Credential Stuffing", <https://owasp.org>.
- [5] OWASP, "Deserialization of untrusted data", [https://www.owasp.org/index.php/Deserialization\\_of\\_untrusted\\_data](https://www.owasp.org/index.php/Deserialization_of_untrusted_data).
- [6] K.Pranathi, S.Kranthi, Dr.A.Srisaila, P.Madhavilatha, "Attacks on Web Application Caused by Cross Site Scripting", 2018 Proceedings of the 2nd International conference on Electronics, Communication and Aerospace Technology
- [7] OWASP, "XSS (Cross Site Scripting) Prevention Cheat Sheet"
- [8] Yahoo, "serialize-javascript", <https://github.com/yahoo/serialize-javascript>
- [9] Wade Alcorn, "BeEF", <https://beefproject.com>
- [10] Justin Samuel, "RequestPolicy", <https://www.requestpolicy.com/index.html>