# Antecedents

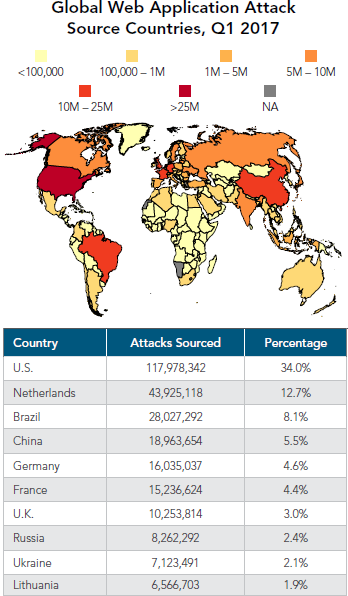
## Background

Computer security has always been a major concern in the field of technology. Humans first tried to purposely exploit vulnerabilities in networks back in the 60s, and then through the 70s, people would develop methods to make free phone calls by replicating the tone required to unlock the AT&T’s phone network. Worms and viruses would first appear years later, created by modifying programs intended to help computers. In the coming decades, self-modifying viruses would be developed, and new methods for massive world-wide distribution would be conceived. Attacks would extend even to government and military computer systems[3][4].

In the mid-2000s, the quick growth of the Internet would only help increase the reach of virus infections thanks to emails and social media, and credit and debit cards would become some of the new targets for attacks[5].

Nowadays, with the advent of the Internet of Things and the growing globalization, the issue of computer security has only grown in importance[6]. Viruses, malware and many other varied threats and attacks can exploit vulnerabilities in a system, requiring appropriate defenses from users and companies in order to avoid information theft or financial harm. Web applications have become a prominent target for attacks due to the multiple vectors of attack that exist and the fact that they tend to hold sensitive or valuable information.

The world has seen a 35% increase in total web application attacks in the first quarter of 2017 compared to 2016. As reference, the United States is both the most targeted country for web application attacks as well as their greater source. The United States accounted for roughly 34% of all web application attacks worldwide in the first quarter of 2017, up from 28% in the previous quarter. The list of most common sources for attacks after the United States consists of Netherlands, Brazil, China, Germany, France, United Kingdom, Russia, Ukraine, and Lithuania[7], as can be seen in the following figure:

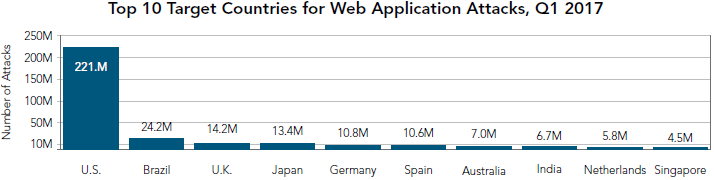


1 Global Web Application Attack Source Countries, Q1 2017

[Source: Akamai’s State of the Internet Security Report]

The Netherlands is the top source for web application attacks in Eurasia, as well as the 2nd worldwide, which is noteworthy given its relatively small population density. In comparison, the United States has nearly twenty times the population but is responsible only for roughly three times the number of attacks than the Netherlands.

As for the targets of the attacks, the United States leads the list as commented above, followed by Brazil, United Kingdom, Japan, Germany, Spain, Australia, India, Netherlands, and Singapore:



2 Top 10 Target Countries for Web Application Attacks, Q1 2017

[Source: Akamai’s State of the Internet Security Report]

As can be seen in the graphic, the attacks targeting the United States account for a vast majority of all attacks (nearly 70%), with the United States receiving almost ten times the number of attacks of the country in second place (Brazil). Even then, the number of attacks targeting the United States is actually down 9% compared to the last quarter. Conversely, Brazil and the United Kingdom have seen an increase in attacks of a 46% and a 30% respectively compared to the last quarter.

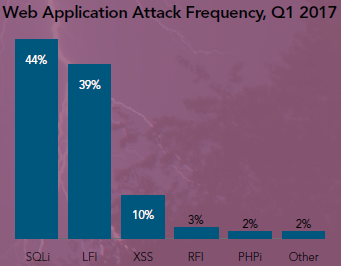
China and Canada, which appeared in this list in the last quarter, have been replaced by Spain and Singapore, both of which have been in this list in the past. These sudden swings may seem significant, but they are within the norms usually expected from such traffic.

## Web Application Attacks

Attacks are techniques used to exploit vulnerabilities in web applications.[[1]](#footnote-1) There exist a wide number of different possible attacks, with varying risks and difficulties in execution. Certain attacks are more susceptible to being automated than others.

According to the OWASP Top Ten Project[[2]](#footnote-2), the greatest security risks as of 2013 are injection, broken authentication and session management, cross-site scripting, insecure direct object references, security misconfiguration, sensitive data exposure, missing function level access control, cross-site request forgery, using components with known vulnerabilities and unvalidated redirects and forwards.

As reference, the web application attacks most frequently used during the first quarter of 2017 were SQL injection, local file inclusion, and cross-site scripting (XSS), followed by remote file inclusion and PHP injection[7]:



3 Web Application Attack Frequency, Q1 2017

[Source: Akamai’s State of the Internet Security Report]

The top three attack vectors have not changed from the last quarter. Their widespread use can be explained by the relative simplicity of their method (they are not as complex as other attack vectors) and their high efficiency. If a website does not actively protect against these vectors, more often than not it will be vulnerable to them.

Some of the most common attacks will be briefly described in the following subsections for clarity, along with some other widespread attacks.

### Injection

Code injection consists on sending untrusted data to an interpreter (SQL, OS, LDAP) as part of a command or query. This might trick the interpreter into executing an arbitrary command in order to access or alter data without proper authorization.

This method is easy to exploit and can make use of diverse interpreter queries (SQL, LDAP, XPath, NoSQL), OS commands, XML parsers, or SMTP headers among others, which makes it one of the most prevalent web application attacks. Some of the most common types of injection are:

-SQL Injection:

This injection attack abuses the input data entry process from the client to the application in order to insert or ‘inject’ SQL commands to dynamically construct a SQL query that will be executed by the target system. By executing SQL arbitrary commands, the attacker might tamper with the target database in multiple ways, such as reading, altering or deleting sensitive data.

-Blind SQL Injection:

This attack is a specific type of SQL injection where the attacker asks true or false questions (Boolean queries) to the database, and then takes advantage of the application’s generic error messages to determine the answer. This method is used by attackers to retrieve information from web applications that don’t output data to the web page.

-XPath Injection / Blind XPath Injection:

The procedure for this attack is identical to that of an SQL injection, except that it’s used for web applications that request user-supplied information to construct an XPath query for XLM data. A blind variant that uses Boolean queries exists.

-Code Injection:

This attack injects code to be directly interpreted by the application, such as PHP code. It exploits a lack of proper input/output data validation.

-Command Injection:

This attack is used to execute arbitrary commands on the host operating system, when a vulnerable application passes user-supplied data such as forms or cookies to a system shell.

### Broken Authentication and Session Management

Authentication and session management functions are often not implemented correctly, presenting flaws in areas such as logout, password management or timeouts. An attacker might abuse these leaks to access passwords, keys or session tokens, or impersonate users.

### Cross-Site Scripting (XSS)

This attack takes advantage of web applications where the output makes use of the user input without validating or encoding it. The attacker might then inject malicious scripts into a trusted web site to create a malicious URL, and use social engineering to trick a victim into visiting such an URL. The user’s browser will assume the script comes from a trusted source and will execute it. This might compromise any sensitive information retained by the browser, such as cookies or session tokens.

-Stored XSS Attacks

Also known as Persistent or Type-I XSS. The script is injected directly into a database, message forum or similar, in a way that it is permanently stored on the target server. When a user requests the stored information, that user will also inadvertently retrieve the malicious script.

-Reflected XSS Attacks

Also known as Non-Persistent or Type-II XSS. The script is sent to the server as part of the input of a request. A vulnerable server might then include the script in the output, such as in an error message or search result. Social engineering is used to lure the victims into clicking a link or submitting a specific form in such a way that the injected script is reflected back to them as an output.

### Insecure Direct Object References

A direct object reference refers to the process of exposing an internal implementation object, such as a file or database key. If incorrectly implemented, any attacker might access unauthorized data by manipulating those references. For instance, without proper access control check, an attacker might access any account by modifying the ‘account’ parameter in their browser’s address.

### Cross-Site Request Forgery (CSRF)

This attack requires the user to be authenticated. Then, making use of social engineering such as sending a link or abusing image tags, the attacker might trick the user into executing an unintended action. Once authenticated, the web application has no way to differentiate between a genuine request and a forged one. While this might not be used to steal data, since the attacker doesn’t see the response to the forged request, it might force the victim to execute actions such as transferring funds or deleting an account.

### Clickjacking

Multiple transparent or opaque layers are used to trick the user into unintentionally clicking on a button or link, which might redirect them to a malicious site or force them to execute an undesired action.

### Denial of Service (DoS)

The attacker floods the target system with a large number of requests, making it unavailable for legitimate users. Alternatively, the attacker may instead exploit a programming vulnerability to achieve the same effect.

### Man-in-the-middle

The attacker intercepts a communication between two systems. For instance, http transactions normally involve a TCP connection between client and server, but an attacker could split that connection into two (client-attacker and attacker-server), allowing the attacker to read or modify the exchanged data.

### Man-in-the-browser

This attack follows the same approach as the man-in-the-middle attack, but it makes use of a Trojan Horse as an interceptor between the browser and its security mechanisms. It bypasses authentication factors.

### Brute Force Attack

The attacker makes requests to a server testing every possible value of a parameter, analyzing the responses until a successful value is found. While the attacker might try every possible combination, it’s common to instead try for specific sets of values such as a dictionary attack or rainbow tables.

### Social Engineering

Social engineering refers to the practice of taking advantage of human psychology in order to access confidential information, instead of exploiting flaws in software[8]. It could be divided into multiple categories:

-Baiting:

The attacker leaves a malware-infected physical device such as a USB flash drive in a place where the victim is likely to find it, usually with a curiosity-inciting label. The unsuspecting victim might pick up the device and plug it into their computer, installing the malware.

-Phishing:

The attacker impersonates a reputed source and sends an email to the victim, trying to trick the recipient into sharing personal or confidential information, or clicking a malicious link. When the target is a specific individual or organization, it receives the name of ‘spear phishing’.

-Pretexting:

The attacker will make use of a pretext (an invented scenario or elaborate lie) to manipulate the victim into divulging confidential information, such as a scammer pretending to need certain information pertaining to their target in order to confirm their identity.

-Scareware:

Scareware is a type of malware that attempts to trick victims into downloading malicious software. Scareware might disguise itself as system messages coming from an antivirus or firewall application, and falsely inform about a number of inexistent problems, suggesting purchasing or downloading actual malicious software to fix the problems.

## Attack Prevention

### Injection

Preventing injection attacks requires the interpreter to be able to make a distinction between untrusted data and actual commands and queries, which can be achieved by one of many ways.

The most obvious solution would be to forgo use of the interpreter altogether and instead use a safe API (Application Programming Interface), or at least an API that provides a parameterized interface.

If APIs can’t be used for this purpose, injection can be prevented using an interpreter and making careful use of escaping routines. Injection attacks will usually involve injecting special characters that are normally not found in legitimate queries (such as ‘<’ or ‘>’), so checking for these special characters might help differentiate between an actual user query and a potential script from an attacker. By escaping these special characters (using the escape syntax for that particular interpreter) or sanitizing the input, it should be possible to block those kinds of attacks.

Another possibility is white list validation, which only takes in data structured in a particular way, such as properly formatted dates or e-mail addresses, and disregards the rest of the input.

### Broken Authentication and Session Management

The most important advice for an organization is to make sure that their developers have a single set of strong authentication and session management controls. It’s also important to prevent XSS flaws that could be abused to steal session IDs.

Authentication credentials should always be hashed or encrypted when stored, and should not be sent over unencrypted connections. Special care must be taken with account management functions such as password recovery, for they could be abused to guess or overwrite credentials.

Session IDs are especially vulnerable. They shouldn’t appear in the URL, they should timeout, and they should be rotated after a successful login. Also, it’s important that any used authentication tokens are properly invalidated during logout.

### Cross-Site Scripting (XSS)

Preventing cross-site scripting requires separating legitimate browser content from untrusted data. The simplest method, effective against both reflected and stored XSS, is to filter data according to its HTML context (body, attribute, URL) and escape any data deemed untrusted.

Similar to injection, white list server-side input validation can also be used, allowing only data that follows a specific format, such as length ranges or valid characters, and disregards everything else.

Content Security Policy (CSP) can be used to restrict from which location and what type of resources is the client browser allowed to load, so it could be used to defend against XSS across an entire webpage.

### Insecure Direct Object References

The simplest defense against insecure direct object references is to establish an appropriate check access control. Whenever an object or service is requested, the application should perform an access control check to confirm whether the user is authorized for the requested object.

Another method is to use an indirect reference map, which would be used for mapping from a set of internal object references to a set of indirect references. This way, the user would only have access to these per-user specific indirect references, which would be sent to the database to retrieve the actual request (the internal object reference), such as a filename or database key.

### Cross-Site Request Forgery (CSRF)

The most effective way to protect against CSRF is to include an unpredictable token in each HTTP request, so that the attacker would be unable to execute a CSRF without knowledge of this token. This token should be unique per session and should preferably be included in a hidden field, so that it is sent in the body of the HTTP request and not in the URL.

Another solution is to require the user to confirm all their transactions. This can be achieved by asking them to reauthenticate or requesting proof that they are an actual user (such as by a CAPTCHA).

## Manual Testing

### Introduction

Before the testing process can be automated, it is necessary to have an understanding of how to manually uncover vulnerabilities. In order to achieve this, a few basic testing examples will be described. They will all be carried out without the assistance of pen-testing tools, using a purposely vulnerable web server.

For the purposes of these examples, a virtual machine hosting an Ubuntu 16.04 system will be used, but the basic procedure is applicable to other systems. The used vulnerable web server will be OWASP Mutillidae 2.6.43.[[3]](#footnote-3) It should be noted that the vulnerabilities found within this server are genuine, and should never be deployed outside a virtual machine.

### Injection

An injection attack requires a field that accepts input data, and manipulating it in order to create an arbitrary query or command. The syntax will change depending on the interpreter. Error messages can help find what type of database the application uses, since many developers allow their applications to display error messages by default.

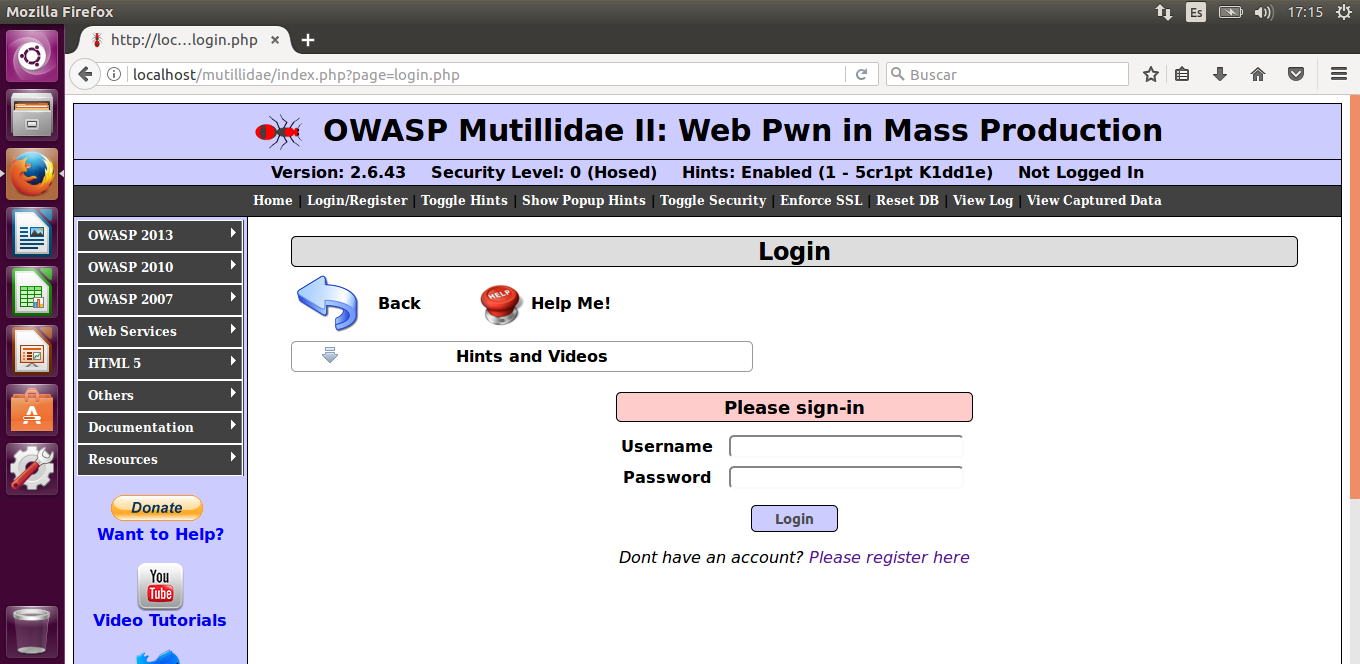
For example, let’s assume the server stores credentials in a MySQL table with this format:

|  |  |  |
| --- | --- | --- |
|  | **Username** | **Password** |
| **1** | User1 | Pass1 |
| **2** | User2 | Pass2 |
| **3** | User3 | Pass3 |

The table might contain any number of additional columns, and the passwords might or might not be encrypted or hashed. In order to retrieve information, the server will use MySQL queries to access the required information. For instance, inputting the username “admin” and the password “adminpass” might construct a query such as:

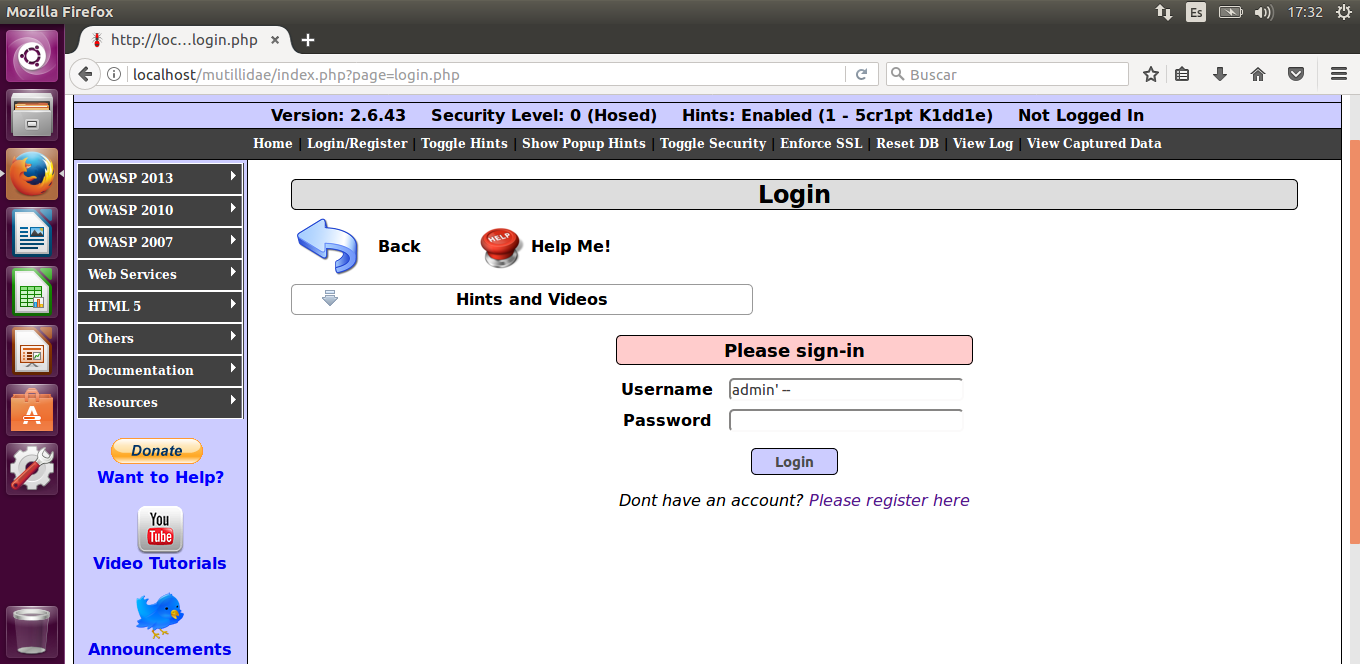
﻿SELECT \* FROM accounts WHERE username='admin' AND password='adminpass'

This will search through the ‘accounts’ table in the database for any row where both the username and the password correspond to the inputted data. The query is thus dynamically constructed, taking user input and sending it to the function as arguments. If the input is not sanitized, it’s possible to abuse this to force the server to execute an arbitrary query. An example of this might be a login page.



4 Standard Login Form

For instance, let’s assume that an attacker wants to log in as ‘admin’ without knowing the password. Since the input is passed on as arguments to the MySQL query, this would allow the attacker to modify the query. Knowing that comments in MySQL are inserted with a double hyphen (--) followed by a space, the attacker might input this in the username field:



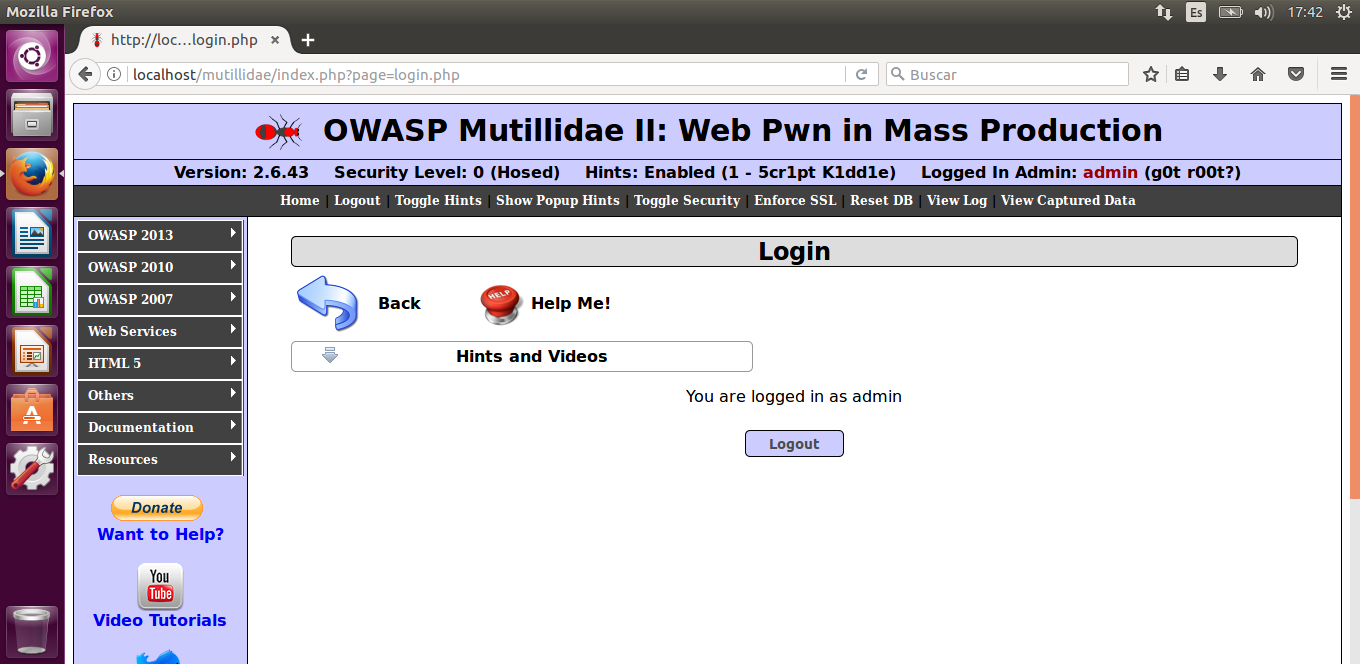
5 Bypassing a Login Form

This will construct the following query:

SELECT \* FROM accounts WHERE username='admin' *-- ' AND password=''* Since everything behind the double hyphen is commented out, this query is equivalent to:

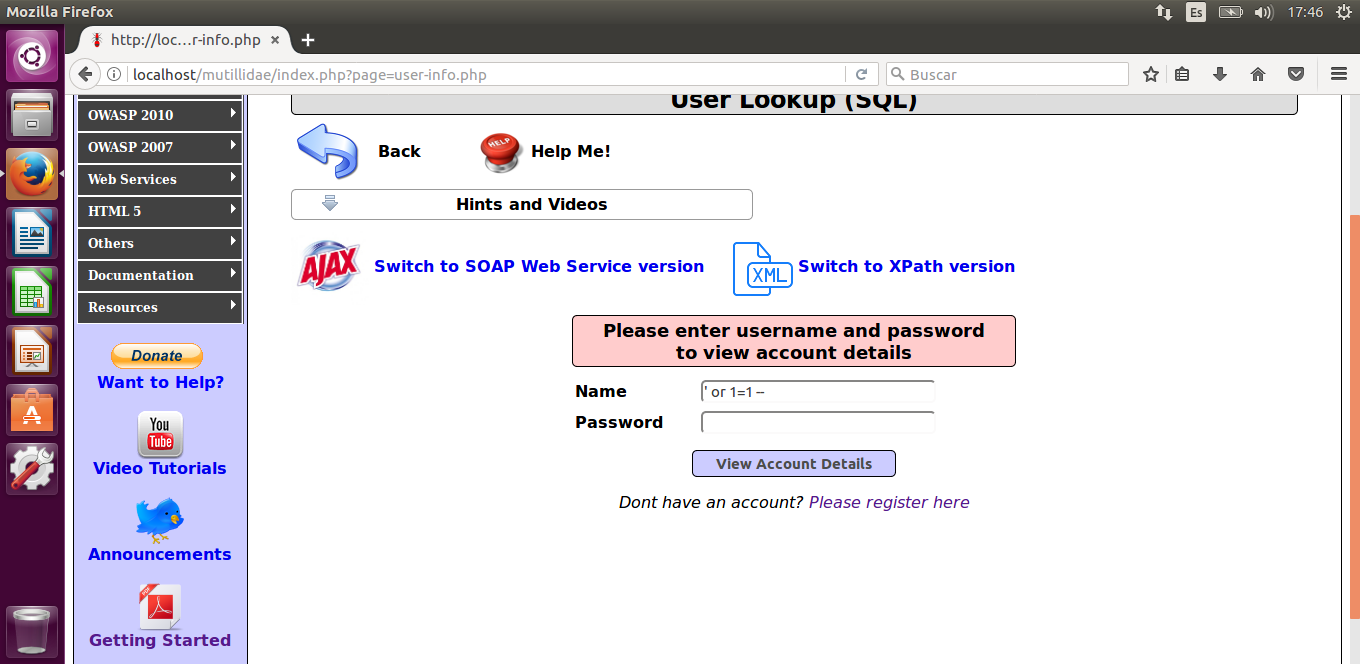
SELECT \* FROM accounts WHERE username='admin'

This query will return the row with the username ‘admin’, regardless of the password, allowing an attacker to bypass authentication.



6 Succesful Login Form Bypass

Expanding on this, it would be possible to force the server to dump the whole database by constructing a query that returns true for every row. For example, in a user lookup page, this input:



7 User Lookup Bypass

Would construct this query:

SELECT \* FROM accounts WHERE username='' OR 1=1 *-- ' AND password='adminpass'*

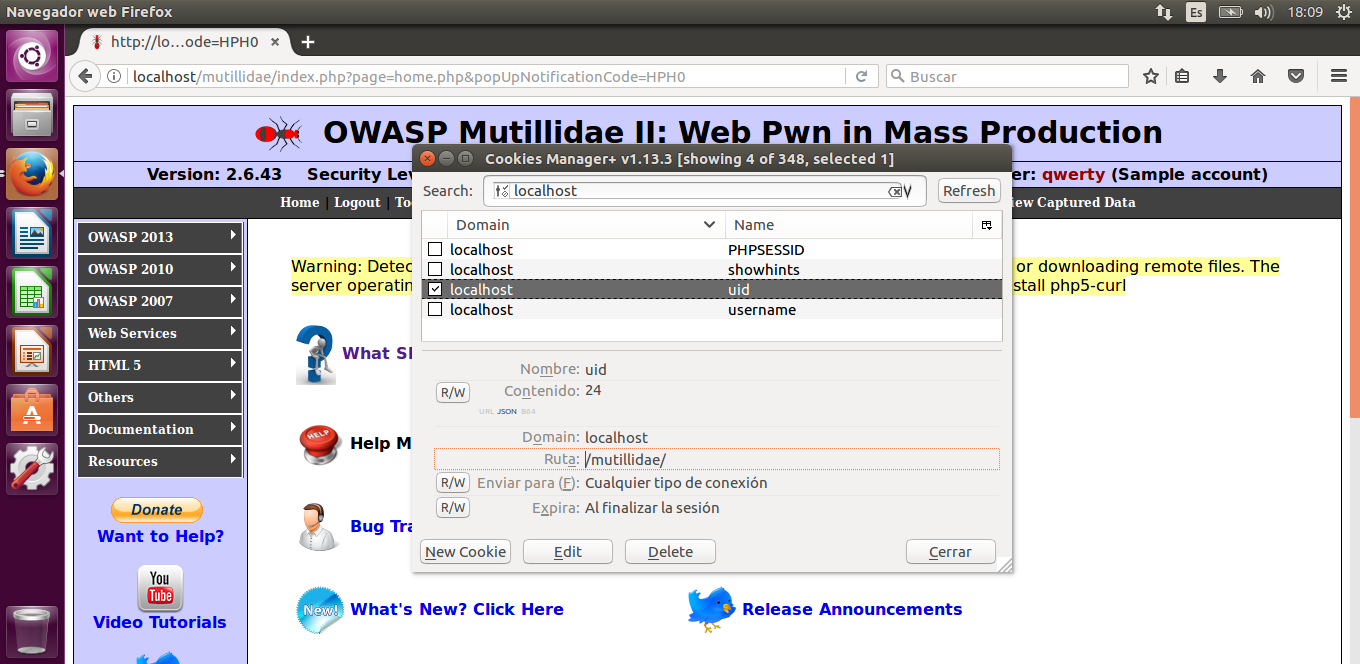
Ignoring the commented out part, this query will return every row where either of the two conditions evaluate to true. By inputting a trivial second condition that will always be true (such as 1=1), this query will return every row in the database, dumping the credentials of every user.

While these examples have been done in MySQL, the methodology is identical for other different interpreters, varying only in the syntax used to construct queries.

### Broken Authentication and Session Management

There are multiple ways to hijack sessions depending on how the particular target web application is configured. As mentioned before, in an insecure page injection can be used to extract password lists or even bypass authentication altogether, but there are other ways to achieve this.

Depending on how a page works with cookies, they could be exploited to obtain administrative privileges or impersonate another user. There are many ways to edit cookies. For the purposes of this example, the ‘Cookies Manager+’ add-on for Mozilla Firefox will be used. First, it is necessary to understand how the target web application manages cookies. An easy way to begin is to actually register as a normal user, in order to make the web application generate a genuine session cookie.



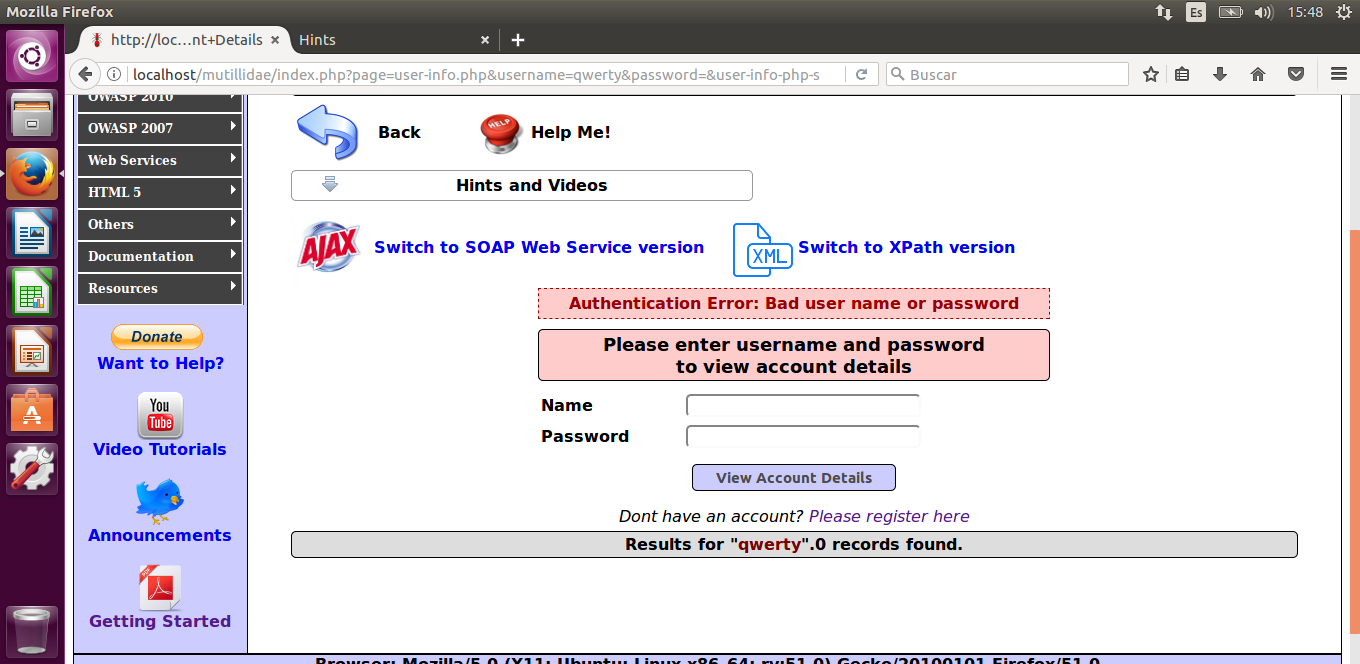
8 Cookie Manipulation

Analyzing the newly generated cookies can give insight on how to proceed. For this particular example, registering as a new user has created two new cookies, ‘uid’ and ‘username’, corresponding to the user’s id and the username. In most cases the relationship between the cookies and the session will not be as obvious, so it might be necessary to create more than one account to compare the cookies and see how they relate to their respective sessions. For this example, trial and error shows that changing ‘uid’ to 1 and ‘username’ to admin is equivalent to logging in as the web’s administrator.

### Cross-Site Scripting (XSS)

Much like injection, cross-site scripting attacks make use of fields that allow the user to input data and then display it back in the output without proper encoding. Two variations exist, reflected XSS and stored XSS.

Reflected cross-site scripting is transient and is not stored within the server. Any page that directly outputs the user input without sanitizing it is vulnerable. For example, let’s assume a user lookup page takes in a username and a password and then displays the username back without sanitizing it. For instance, inputting ‘qwerty’ would return:

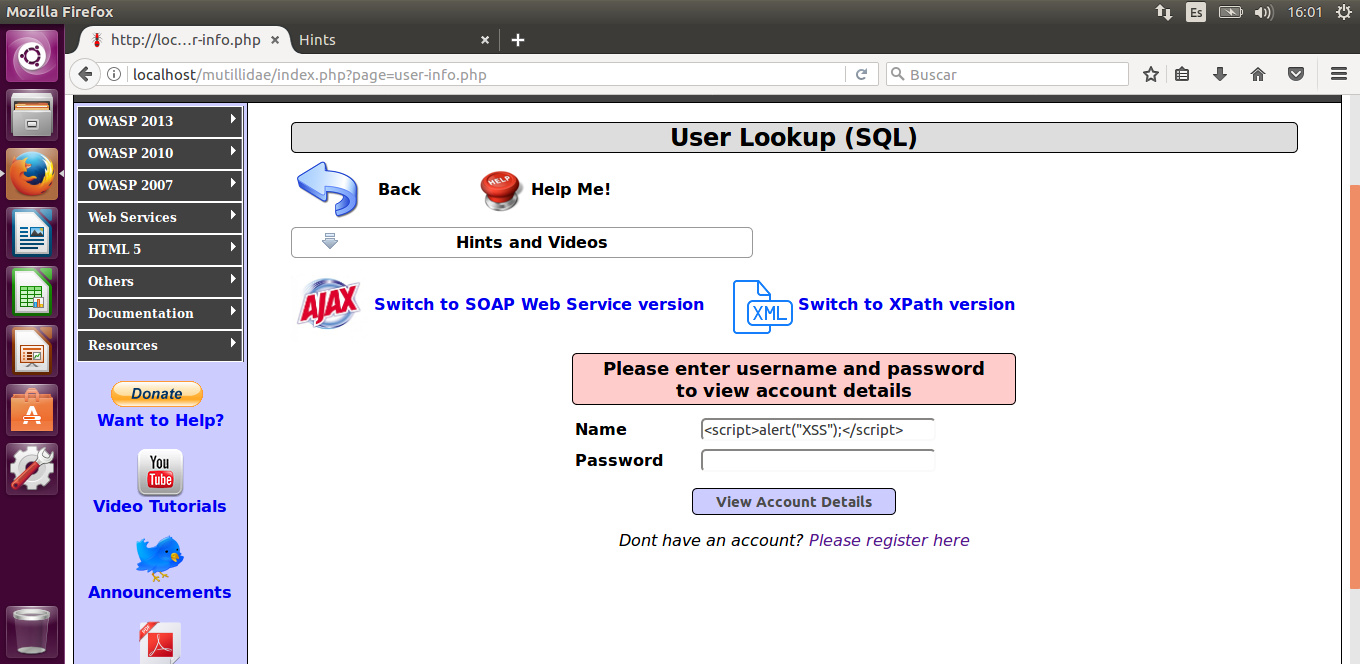


9 Login Error Example

If the developer is careless and passes on the parameters directly through the URL, the URL might look like this:

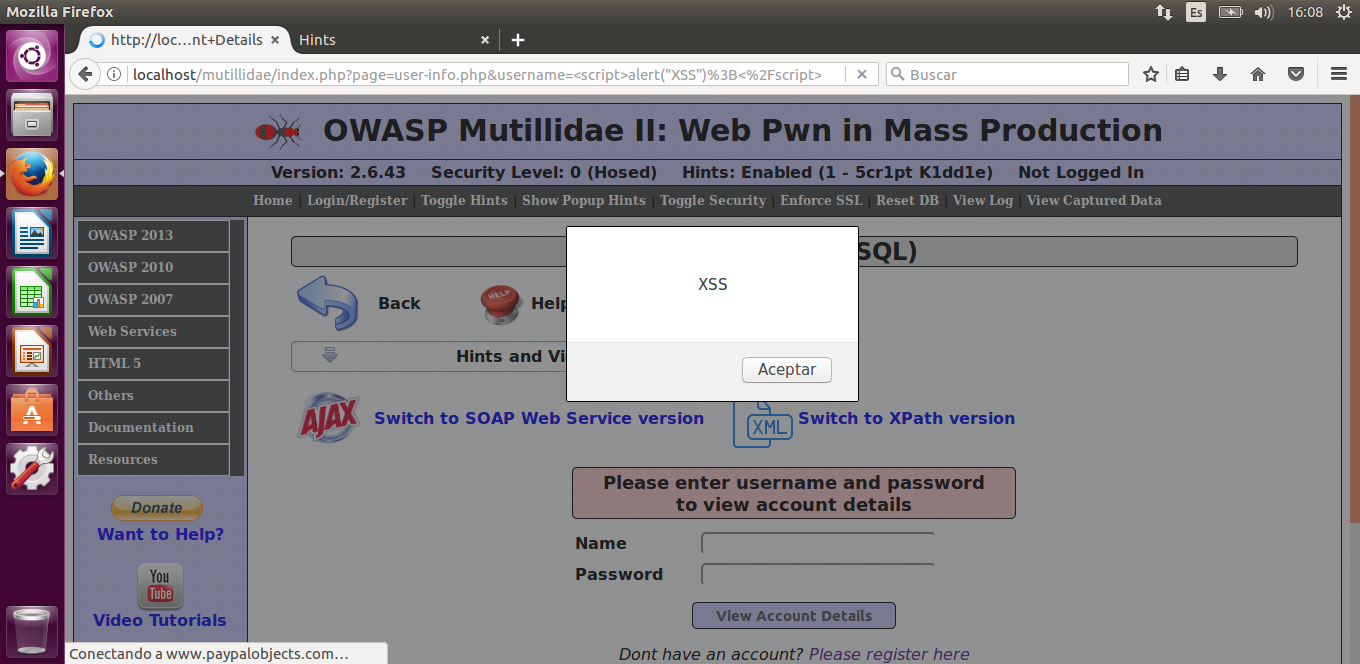
*…/mutillidae/index.php?page=user-info.php&****username=qwerty****&password=123&user-info-php-submit-button=View+Account+Details*

If that is the case, it would be possible to input a malicious script in the username field in order to generate a malicious link that will execute that script when accessed. The attacker could then make use of social engineering to trick a victim into clicking such a link and executing the script. A very simple example of this could be a script that displays a warning message:



10 Reflected XSS Example

This would generate the URL: *…/mutillidae/index.php?page=user-info.php&****username=%3Cscript%3Ealert%28%22XSS%22%29%3B%3C%2Fscript%3E****&password=&user-info-php-submit-button=View+Account+Details*. The apparent garbling in the URL happens because all non-alphanumeric characters are substituted by their equivalent ASCII codes (such as ‘<’ becoming %3C). Upon accessing this URL, the script is executed and the warning message is displayed.

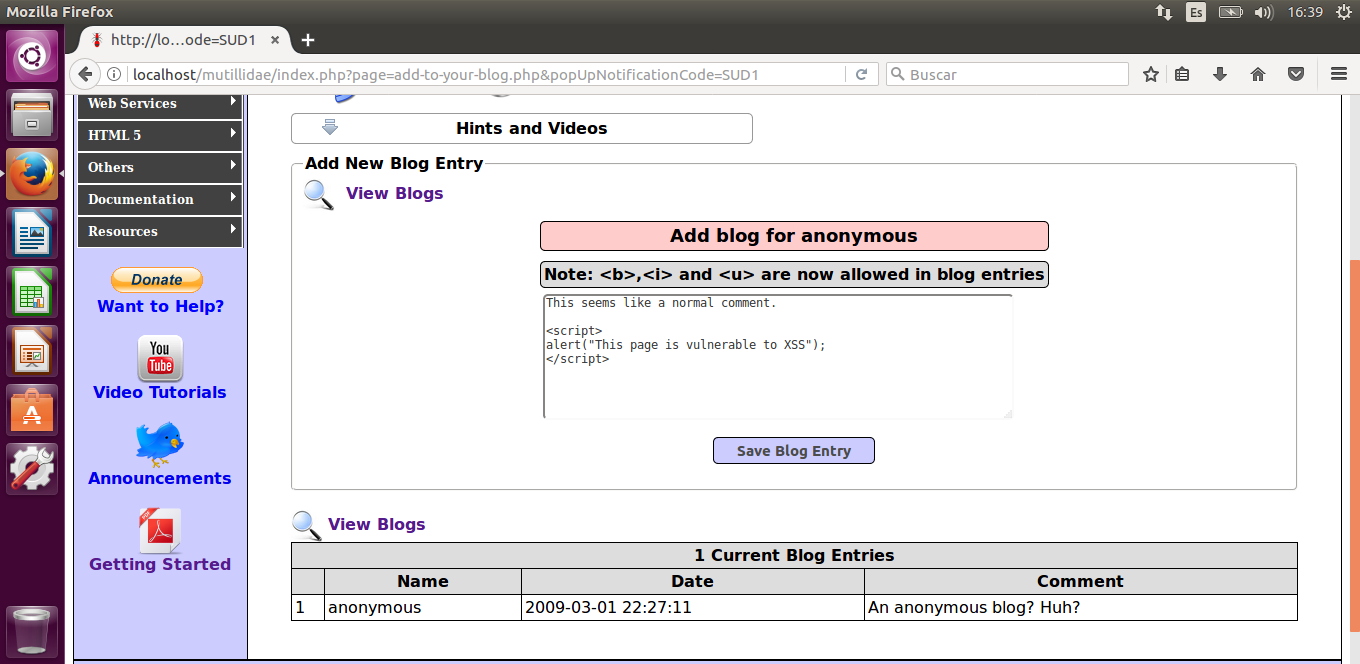


11 Succesful Reflected XSS

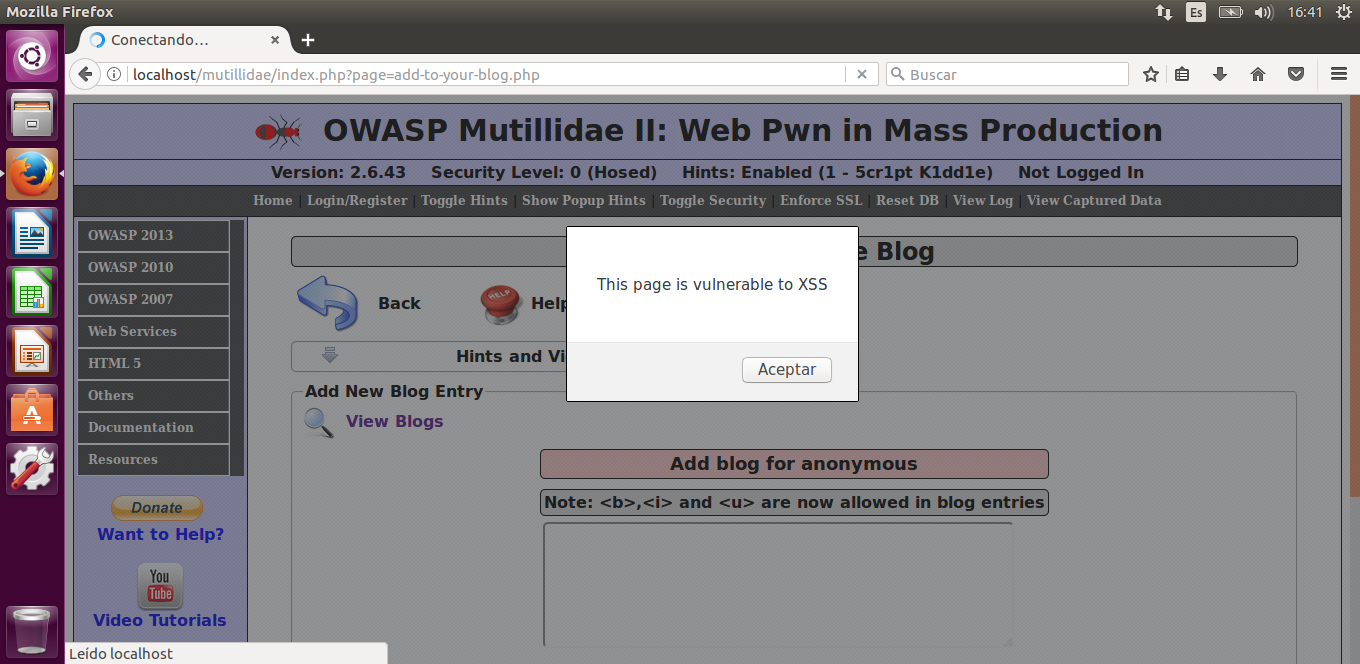
While this particular example is benign, an attacker could force the execution of any arbitrary script of their choice using this method.

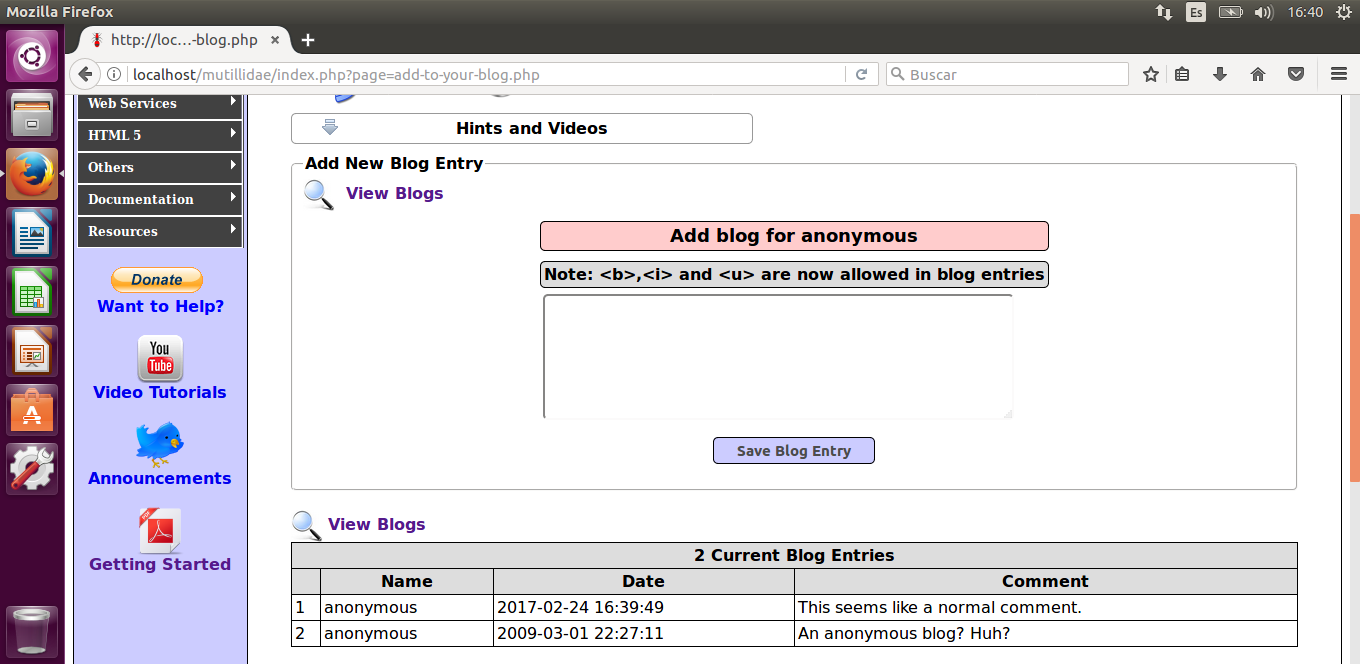
Stored cross-site scripting differs from the reflected variant in that it doesn’t need to target a specific victim. Instead, as its name implies, the script is directly stored in the server and executed whenever any user accesses that page or performs a specific action. Pages vulnerable to stored XSS include all those that allow users to input data that is stored in the server without sanitizing it first. Common examples are forums, blogs or guestbooks.

An example could be a blog that allows users to leave comments that are then displayed to every other visitor. If the inputted comment was a malicious script, every visitor would accidentally execute it upon visiting the page.



12 Stored XSS Example





13 Succesful Stored XSS

This entry would be added as a permanent comment, and in most cases it would look no different than a normal comment, but it would execute the script whenever a user visits the page. Subtler scripts could execute without the user noticing anything wrong. The only way to solve this would be for the webmaster to delete the malicious entry from the server. In particularly pernicious cases with self-replicating scripts, the only solution might be to restore the database to a previous backup or reset it altogether.

### Insecure Direct Object References

Insecure direct object references will take place whenever an application tries to access a resource using the resource itself as the reference to look up for. Through trial and error, it could then be possible to gain access to a confidential file without having the required privileges.

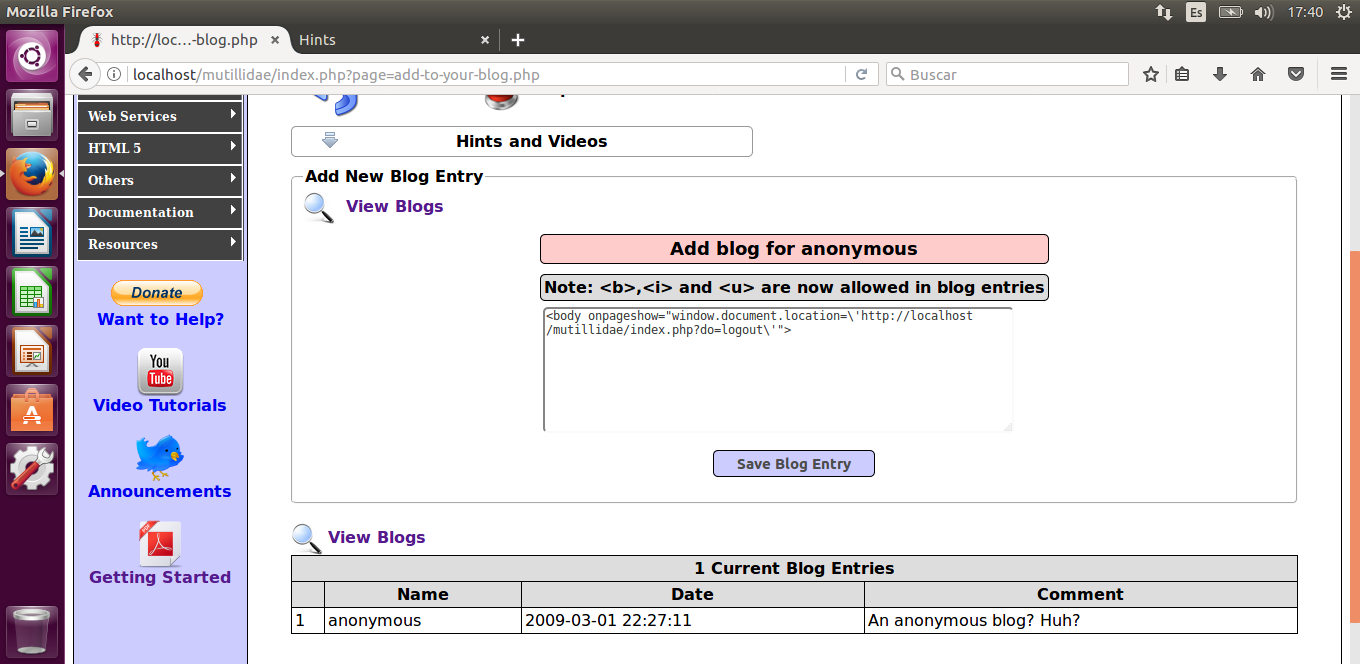
If the web application has a folder called ‘passwords’ that contains a text file of the name ‘accounts.txt’, it could be possible to access it directly through the URL */passwords/accounts.txt* ,even if the text file itself isn’t accessible through normal means in the webpage. Another example would be a page that shows the user’s account directly through the URL as a parameter, allowing attackers to just modify that parameter to access any account, such as */userInfo?account=****admin***.

### Cross-Site Request Forgery (CSRF)

Before this kind of attack can take place, the target must be authenticated. Then, the attacker runs a script into the target’s browser to forge a request. The website has no way to know that the request isn’t genuine, since it comes from the user’s browser.

Cross-site request forgery can happen in multiple ways, and can be combined with many of the previous attacks. The final objective is to make the target’s browser execute a script, and this can be achieved by many different ways such as sending an URL or XSS.

As an example, in a forum or comment page that doesn’t sanitize input an attacker might use cross-site scripting to post an entry with the following HTML code that forces any visitor to logout:



14 CSRF Example

These kinds of attacks are not effective if the user isn’t authenticated beforehand. Other possible approaches for attacks could include forcing a user to make a purchase or sending money to the attacker’s account.

1. https://www.owasp.org/index.php/Category:Attack [↑](#footnote-ref-1)
2. https://www.owasp.org/index.php/Top10#OWASP\_Top\_10\_for\_2013 [↑](#footnote-ref-2)
3. https://www.owasp.org/index.php/OWASP\_Mutillidae\_2\_Project [↑](#footnote-ref-3)