Café Roza

- Cyber Security Project -



**Authors**

Ivan Yulin 320822505

Evgeni Malinsky 311873400

**Lecturer**

Omri Sagron

Table of Contents

Executive summary

Introduction

Acquired knowledge

Vulnerabilities and secure implementations

* SQL Injection
* Broken Authentication
* Reflected XSS
* Stored XSS
* Sensitive Data Exposure
* Insecure Encryption
* URL Restriction
* Redirect Hijack
* Browse Automation with Selenium
* Bypassing input validation

Conclusions

Executive Summary

The main goal of our project is to show secure implementations against most common and most exploited cyber-attack in web applications.

Due to rising cyber-attacks in the world, and a stunning ignorance about it from the beginner programmers, our main intent was to prepare a project that can be used for teaching future cyber security students.

For that cause we have created two (almost) identical websites:

The first one is a website with a “simple and easy” implementation without considering cyber-attacks.

This version is prone to at least 10 popular attacks.

The second version is a website that looks exactly the same where we offer secure implementations for these attacks.

After presenting each vulnerability and its potential consequences in the first version, we offer a correct and secure implementation from the second version.

We hope that our work contribute to future students of cyber security.

Introduction

Web security, your site and your network

Web sites are unfortunately prone to security risks. And so are any networks to which web servers are connected. Setting aside risks created by employee use or misuse of network resources, your web server and the site it hosts present your most serious sources of security risk.

Web servers by design open a window between your network and the world. The care taken with server maintenance, web application updates and your web site coding will define the size of that window.

### Web security risk - should we be worried?

If you have assets of importance or if anything about your site puts you in the public spotlight then your web security will be tested.

It's well known that poorly written software creates security issues. The number of bugs that could create web security issues is directly proportional to the size and complexity of your web applications and web server. Basically, all complex programs either have bugs or at the very, least weaknesses. On top of that, web servers are inherently complex programs. Web sites are themselves complex and intentionally invite ever greater interaction with the public. And so the opportunities for security holes are many and growing.

Technically, the very same programming that increases the value of a web site, namely interaction with visitors, also allows scripts or SQL commands to be executed on your web and database servers in response to visitor requests. Any web-based form or script installed at your site may have weaknesses or outright bugs and every such issue presents a web security risk.

Contrary to common knowledge the balance between allowing web site visitors some access to your corporate resources through a web site and keeping unwanted visitors out of your network is a delicate one. There is no one setting, no single switch to throw that sets the security hurdle at the proper level. There are dozens of settings if not hundreds in a web server alone, and then each service, application and open port on the server adds another layer of settings.

A web security issue is faced by site visitors as well. A common web site attack involves the silent and concealed installation of code that will exploit the browsers of visitors. Your site is not the end target at all in these attacks. There are, at this time, many thousands of web sites out there that have been compromised. The owners have no idea that anything has been added to their sites and that their visitors are at risk. In the meantime visitors are being subject to attack and successful attacks are installing nasty code onto the visitor's computers.

### Known web security vulnerabilities and unknown vulnerabilities

As you know there are a lot of people out there who call themselves hackers. You can also easily guess that they are not all equally skilled. As a matter of fact, the vast majority of them are simply copycats. They read about a KNOWN technique that was devised by someone else and they use it to break into a site that is interesting to them, often just to see if they can do it. Naturally once they have done that they will take advantage of the site weakness to do malicious harm, plant something or steal something.

A very small number of hackers are actually capable of discovering a new way to overcome web security obstacles. Given the work being done by tens of thousands of programmers worldwide to improve security, it is not easy to discover a brand new method of attack. Hundreds, sometimes thousands of man-hours might be put into developing a new exploit. This is sometimes done by individuals, but just as often is done by teams supported by organized crime. In either case they want to maximize their return on this investment in time and energy and so they will very quietly focus on relatively few, very valuable corporate or governmental assets. Until their new technique is actually discovered, it is considered UNKNOWN.

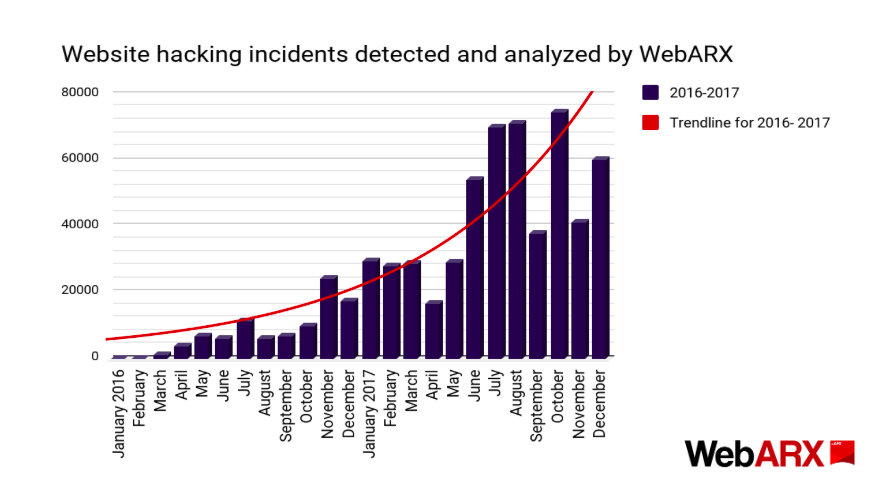
## The number of hacked sites rises rapidly

“There was a 32 percent increase in the number of hacked sites in 2016 compared to 2015.”

– Google Webmaster Central Blog

In March 2016 Google announced that **more than 50 million websites** worldwide are infected or malicious.   
In March 2015, that number was 17 million.

The number of new Web application vulnerabilities published **in 2017 was 212% greater than the number disclosed in 2016.**

**There were detected 95 787 hacked sites in 2016.**   
**In 2017 that number was almost six times higher**

Acquired Knowledge

Before beginning the implementation of our website, we had to acquire a lot of knowledge from different fields.  
some of the things we had to learn and understand:

* Different kinds of hackers and their motivations.

We had to understand where and how would a common attacker try to exploit a website,

Thus understanding even deeply the different kinds of attacks.

* Server side, client side and the difference between the two.
* How to use and work with a remote data base.
* How to set up a server and the differences between a local host, a public host and a sub domain.
* How to use different hacking tools such as: Burpee, sqlmap, Silenium and of course the browser developer tools.
* Coding languages that we used to build a web application:
* HTML - Hypertext Markup Language is the standard markup language for creating web pages and web applications. With Cascading Style Sheets and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web
* CSS - Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.
* PHP - Hypertext Preprocessor is a server-side scripting language designed for Web development, but also used as a general-purpose programming language. It was originally created by Rasmus Lerdorf in 1994, the PHP reference implementation is now produced by The PHP Group.
* SQL - a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.
* Python - an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.
* Technologies and programs that improved our work on the project and the final result:
* Bootstrap - a free and open-source front-end framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.
* jQuery - a JavaScript library designed to simplify the client-side scripting of HTML. It is free, open-source software using the permissive MIT License. Web analysis indicates that it is the most widely deployed JavaScript library by a large margin
* Selenium - a portable software-testing framework for web applications. Selenium provides a playback tool for authoring tests without the need to learn a test scripting language.
* Github - a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features.
* Sqlmap - an open source software that is used to detect and exploit database vulnerabilities and provides options for injecting malicious codes into them. It is a penetration testing tool that automates the process of detecting and exploiting SQL injection flaws providing its user interface in the terminal.

Vulnerabilities and Secure Implementations

SQL Injection

SQL injection is currently the most common form of web site attack. This might be because web forms are very common, often not coded properly and the hacking tools used to find weaknesses and take advantage of them are commonly available online. This kind of exploit is easy enough to accomplish that even inexperienced hackers can accomplish mischief. However, in the hands of the very skilled hacker, a web code weakness can reveal root level access of web servers and from there attacks on other networked servers can be accomplished.

Structured Query Language (SQL) is the nearly universal language of databases that allows the storage, manipulation, and retrieval of data. Databases that use SQL include MS SQL Server, MySQL, Oracle, Access and Filemaker Pro and these databases are equally subject to SQL injection attack.

Web based forms must allow some access to your database to allow entry of data and a response, so this kind of attack bypasses firewalls and endpoint defenses. Any web form, even a simple logon form or search box, might provide access to your data by means of SQL injection if coded incorrectly.

This is done by entering SQL commands into your form fields instead of the expected data. Improperly coded forms will allow a hacker to use them as an entry point to your database at which point the data in the database may become visible and access to other databases on the same server or other servers in the network may be possible.

Web site features such as contact forms, logon pages, support requests, search functions, feedback fields, shopping carts and even the functions that deliver dynamic web page content, are all susceptible to SQL injection attack because the very fields presented for visitor use MUST allow at least some SQL commands to pass through directly to the database.

In our case, we used the most powerful tool we know for sql injection testing: SQLMap .

SQLmap

Sqlmap is an open source software that is used to detect and exploit database vulnerabilities and provides options for injecting malicious codes into them.

It is a penetration testing tool that automates the process of detecting and exploiting SQL injection flaws providing its user interface in the terminal.

In addition to mapping and detecting vulnerabilities, the software enables access to the database, editing and deleting data, and viewing data in tables such as users, passwords, backups, phone numbers, e-mail addresses, credit cards and other confidential and sensitive information.

Sqlmap has full support for multiple [DBMSs](https://en.wikipedia.org/wiki/DBMS), including [MySQL](https://en.wikipedia.org/wiki/MySQL), [Oracle](https://en.wikipedia.org/wiki/Oracle_Database), [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL), [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server), [Microsoft Access](https://en.wikipedia.org/wiki/Microsoft_Access), [IBM DB2](https://en.wikipedia.org/wiki/IBM_DB2), [SQLite](https://en.wikipedia.org/wiki/SQLite), [Firebird](https://en.wikipedia.org/wiki/Firebird_(database_server)) and SAP MaxDB.

How to set up SQLMap:

Basically, all the explanations are perfectly clear in their official website: <http://sqlmap.org/> so we would explain the steps as simply and as shortly as we can here.

To download SQLMap: click on “download zip file” or “ download tar.gz file”.

You can also install it from the CLI :

It is possible to clone their git repository with the command:

git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev

more specifically:

wget 'https://github.com/sqlmapproject/sqlmap/tarball/master' --output-document=sqlmap.tar.gz

This command will download latest sqlmap package from github into your current directory.

Now run this command to extract your sqlmap package from tar file.

tar -xvf sqlmap.tar.gz

If all done successfully go to to your sqlmap directory and run below command to test sqlmap in ubuntu.

cd sqlmapproject-sqlmap-c4f9e66/

python sqlmap.py --version

Now let the Hack begin.

The standard use of the software in the Unix CLI:

sqlmap -u "http://172.16.0.0/files/file.php?id=1"

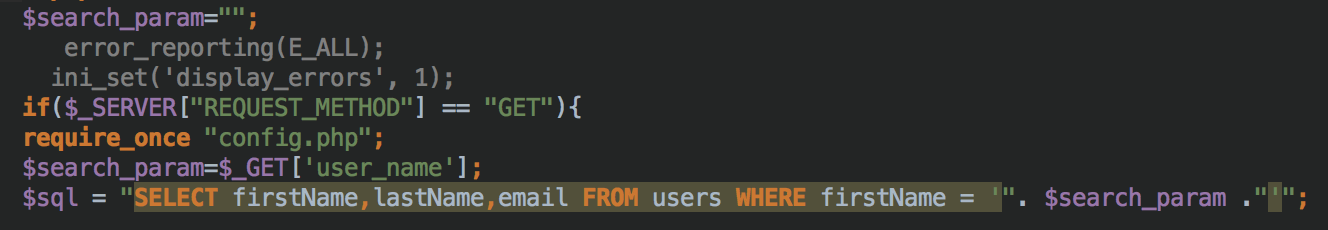
Additional values can be combined:

* --dbs: will display the databases.
* --tables: will display tables in the database.
* --columns: Will display columns in the database.
* --dump: Will dump DBMS database entries.

In our case, we have an sql query that makes a direct access to our database in the Search page.

We had the need to search for something from the database. we made a search from a data table we already have, the users table.

The code looks as follows:



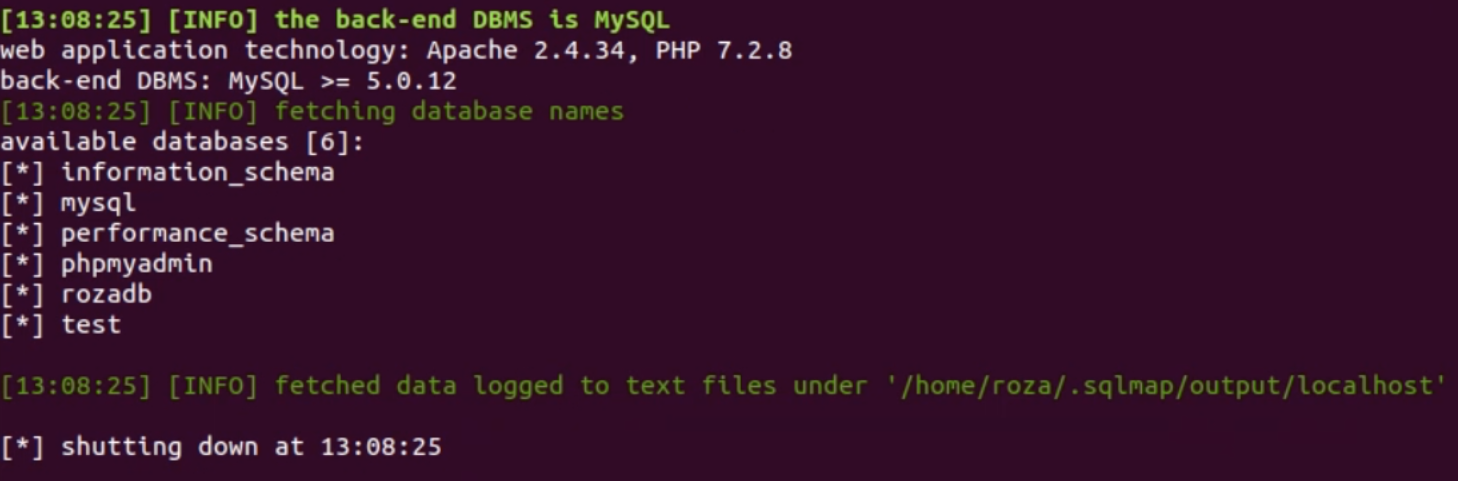
Attack scenario:

* 1. Install SQLmap for your OS from <http://sqlmap.org/>
  2. As shown above, SQLmap gets url as an input. This must be a url that makes a direct access to the database.
  3. In our case, for the Search page we add the next appendix to our url: `search.php` .  
     If a user submits a search he can see an argument at the end of the url: `user\_name` .  
     Thus, the full sqlmap command will look as follows:

sqlmap -u “<http://rozac.byethost7.com/search.php?user_name=>” --dbs

* 1. At this point SQLmap penetrates to the databse through given link.

In our case the result looks like this:



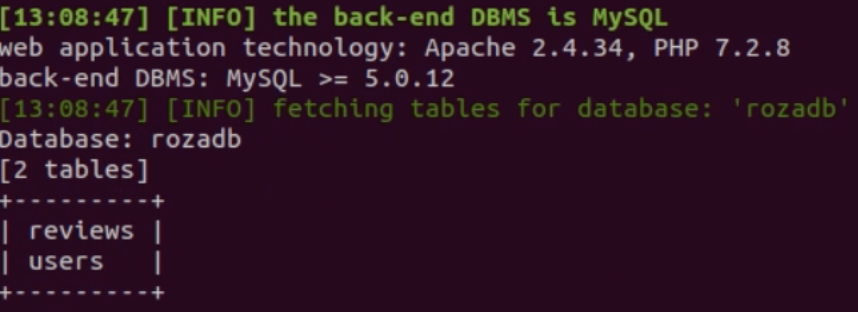
You can see that SQLmap find even the versions of our Apache and PHP .

Below that SQLmap could get all our tables on our databse!

* 1. If we wish to get down to the data itself we could then use a command as follows:

sqlmap -u "http://rozac.byethost7.com/search.php?user\_name= " --tables -D rozadb

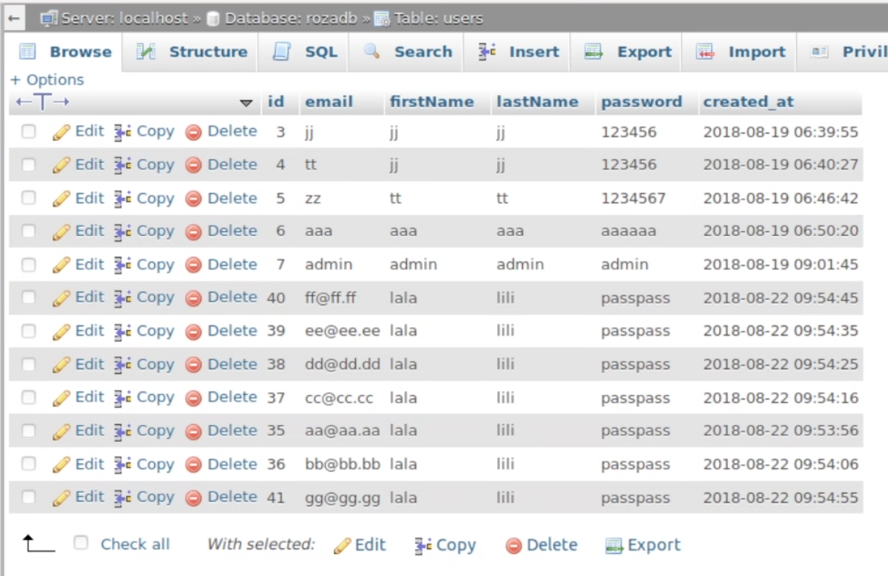
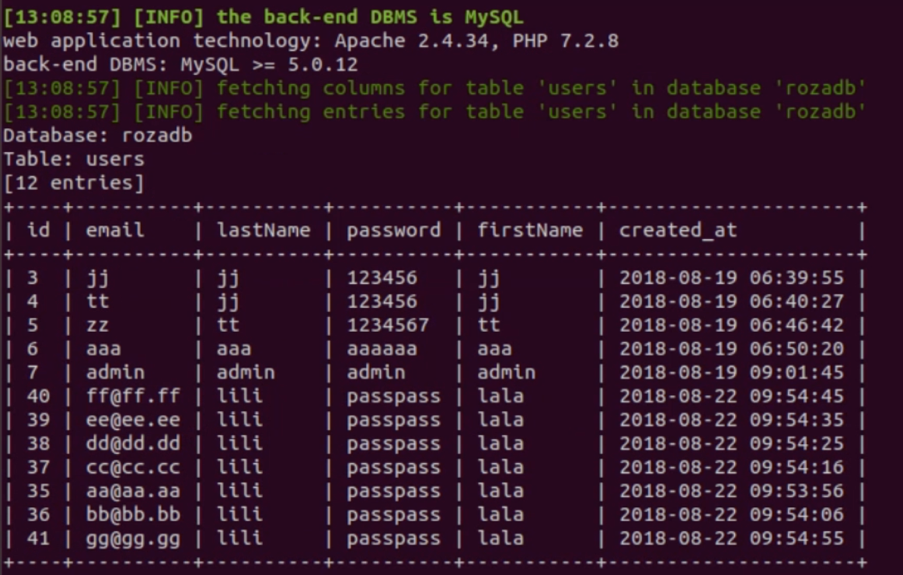
this would show us what’s inside the rozadb table:

Now let’s say we want to go even deeper and get the data we save about our users.

The command will be as follows:

sqlmap -u "http://rozac.byethost7.com/search.php?user\_name= " –dump -D rozadb -T users

the result:

all the data we save about our users.

Indeed, you can see that the data shown is **correct**, and that it shows **all** the data.

In conclusion, SQLmap may scan and return everything the database has if you let it penetrate you website.

Secure implementation

We use as much security layers as we can in our back side code:

**if** ($\_SERVER["REQUEST\_METHOD"] == "GET") {  
 **require\_once** "config.php";  
 $sql = "SELECT firstName,lastName,email FROM users WHERE firstName = ?";  
 **if** ($stmt = mysqli\_prepare($link, $sql)) {  
 // Bind variables to the prepared statement as parameters  
 mysqli\_stmt\_bind\_param($stmt, "s", $param\_name);  
 // Set parameters, never trust user input-get/post  
 $param\_name = filter\_input(*INPUT\_GET*, "user\_name", *FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS*);  
 // Attempt to execute the prepared statement  
 **if** (mysqli\_stmt\_execute($stmt)) {  
 // Store result  
 mysqli\_stmt\_store\_result($stmt);  
 // Check if email exists, if yes then verify password  
 **if** (mysqli\_stmt\_num\_rows($stmt) >= 1) {  
 // Bind result variables  
 mysqli\_stmt\_bind\_result($stmt, $firstName, $lastName, $email);

The first difference you can notice is that we do not paste the search name directly in the sql query but we use the php built-in function ` mysqli\_stmt\_bind\_param ` .

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

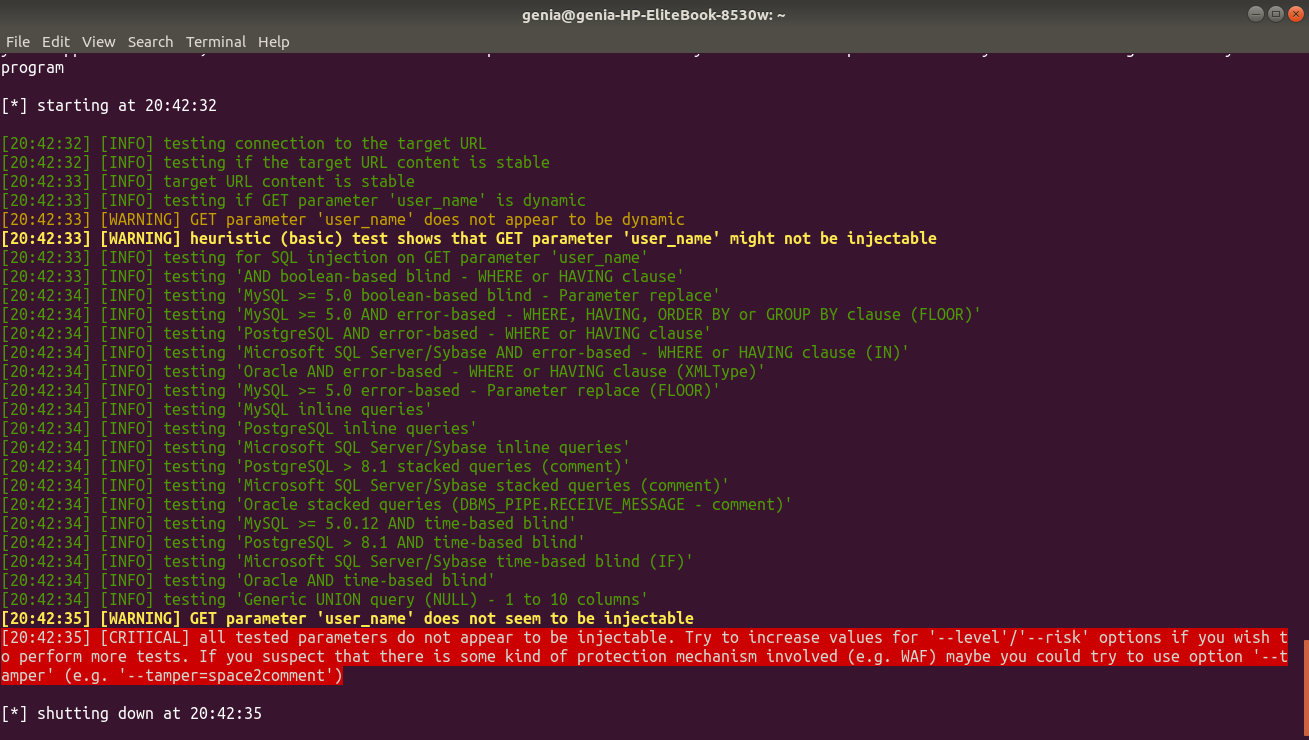
Compared to executing SQL statements directly, prepared statements have three main advantages:

* Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
* Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
* Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

The filter in filter\_input will filter the user\_name with *FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS*  , thus secure the sql query.

*FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS* itself is a php built in function to filter and sanitize input. The best part is that this function will update in case of a security need.

When we try to penetrate the secure version with SQLMap we get the following results:



Broken Authentication

### Most Passwords Can Be Easily Cracked

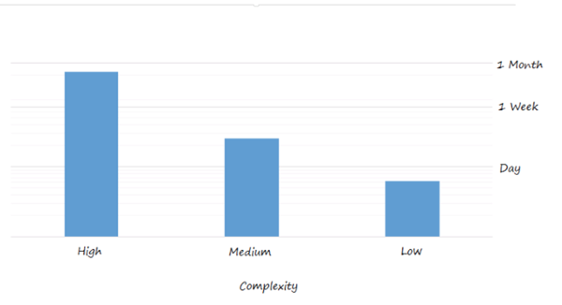
We tried to compute how much time would it take to crack a password with brute force, using standard off-the-shelf cracking hardware (assuming MD5 which is still widely used, standard cracking software [like [this](http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/) : [http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/]](http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/) doing a modest 25B hash/sec).



We then created three password models:

1. Low Complexity - only password length is enforced.
2. Medium Complexity - password length and complexity is enforced. Users have  common ULSD patterns (e.g initial letter is capitalized, last letter is a digit).
3. High Complexity - same as medium complexity, but users are aware not to use common ULSD patterns.

Time required to crack passwords (10 characters) using standard hardware



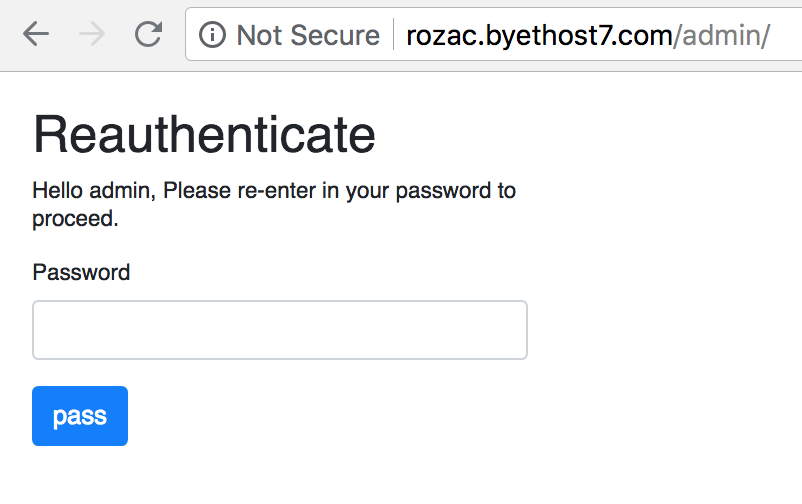
As can be seen, results are astounding: Low complexity passwords can be cracked in less than a day, medium complexity passwords are cracked in less than a week and high complexity password are cracked in less than a month.

Here are some facts we’ve learned:

* **Password complexity isn't working** - passwords can meet complexity and still be considered weak because of password dictionaries.
* **Passwords are not unique** - people reuse passwords and newly leaked dictionaries contain previously leaked passwords.
* **Passwords follow patterns** - in most cases, the top 100 patterns will crack the majority of passwords in an organization.
* **Password cracking is easy** - depending on hardware resources, it can take only seconds to minutes to brute force most passwords.
* **Passwords are shared between users** - people share passwords, use identical passwords and duplicate passwords between services.
* **Password expiration policy is not** enforced - frequent password change policies are disabled, and many times specifically for executives (e.g. CEO) with highly sensitive profiles.

What can we do?

1. Use a password policy to enforce complexity and password expiration. (Note: Preempt can help force a password change when there is an indication that a password was compromised or is considered weak.)
2. Require longer passwords (8 bad, 10 ok, 12+ good)
3. Educate people to:
   1. Not share passwords with other employees
   2. Not share passwords with other cloud services
   3. Not use simple patterns, personal data  or common words (make it unpredictable)
   4. Not repeat passwords when a password expires (enumeration included)
4. Add additional factors to authenticate users. For example, on suspicious logins, you could send end users a simple email notification or push an immediate notification to their mobile device. (Preempt can help in such cases.)
5. Use 2-step verification login process
6. Limit log in attempts – we can understand when a brute force software it doing too many attempts. If we limit the unsuccessful log in attempts to a small number it would increase drastically (up to decades) the time it takes for the software to find a password.
7. Require reauthentication on attempt to access sensitive data.  
   In our case, if a hacker looks up at the ***robots.txt*** file and adds the `/admin` extension to the end of the url he then will be redirected to a webpage that asks for a reauthentication:

we offer the below solution on the server side:

**if** ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  
 // Check if password is empty  
 **if** (**empty**(trim($\_POST["password"]))) {  
 $password\_err = "Please enter your password.";  
 } **else** {  
 $password = trim($\_POST["password"]);  
 **if** ($\_SESSION['loggedin'] == 'true' && $password == $admin\_pass && $\_SESSION["id"] = "7") {  
 header('Location: ' . "userstable.php");

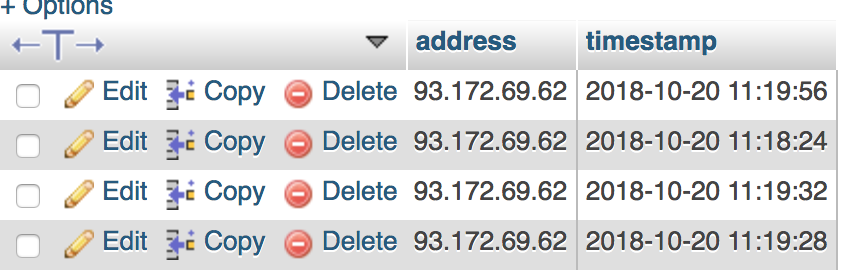
This code runs on the secured version. When entering the admin page this script will ask for reauthentication and allows access to userstable.php only for the admin (id 7) user and only if he was already logged in .

**Limit log in attempts** – we can understand when a brute force software it doing too many attempts. If we limit the unsuccessful log in attempts to a small number it would increase drastically (**up to decades**) the time it takes for the software to crack a password

//check login attempts:  
$ip = $\_SERVER["REMOTE\_ADDR"];  
mysqli\_query($link, "INSERT INTO `ip` (`address` ,`timestamp`)VALUES ('$ip',*CURRENT\_TIMESTAMP*)");  
$result = mysqli\_query($link, "SELECT *COUNT*(\*) FROM `ip` WHERE `address` LIKE '$ip' AND `timestamp` > (now() - interval 10 minute)");  
$count = mysqli\_fetch\_array($result, *MYSQLI\_NUM*);  
  
**if** ($count[0] > 3) {  
 $email\_err = "Your are allowed 3 attempts in 10 minutes";  
}

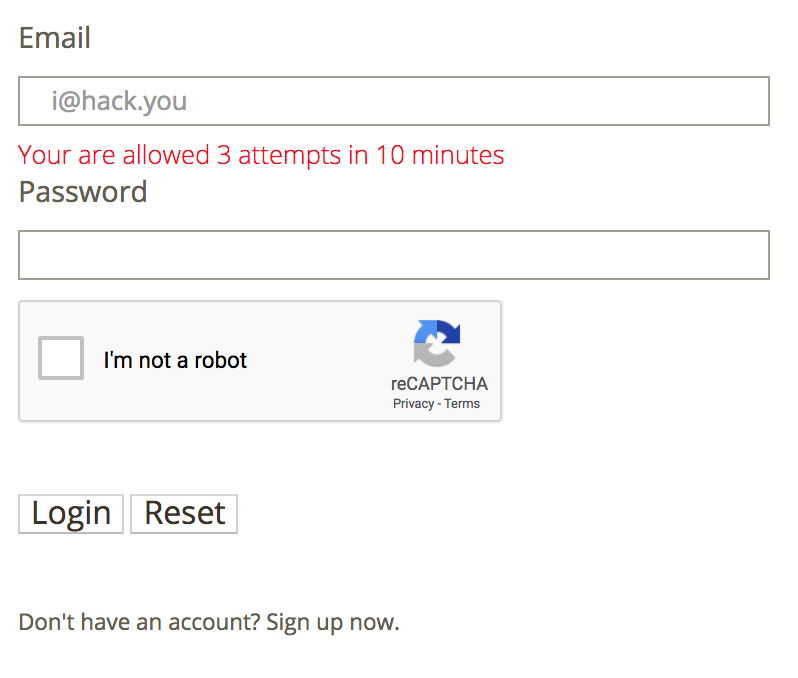
This code runs on the secure version.  
Every attempt from any ip address is saved to the database.   
On each login submition we check wether the amount of login attempts was no more than the limit we set.   
We allow 3 attempts in 10 minutes.

To save memory, the table is reset every 10 minutes



This is how the table looks like when a user makes more than 3 attempts to log in.

Below protection is implemented for the registration page as well.



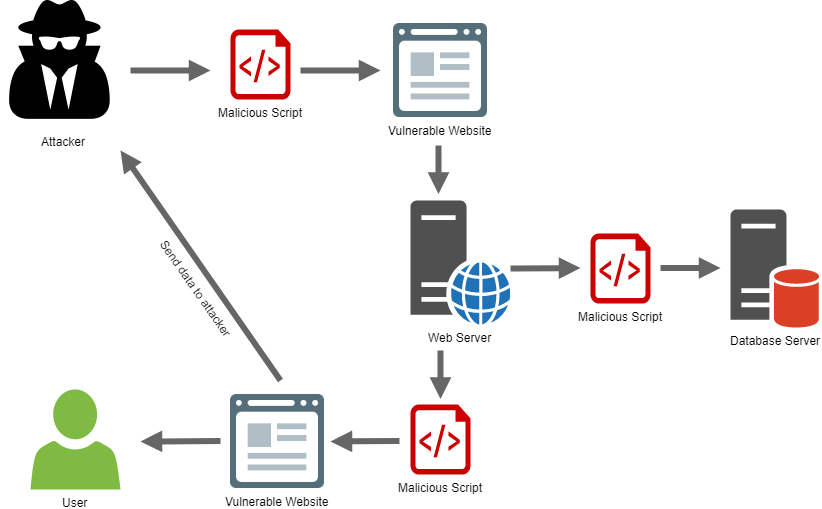
Once a user failed to login 3 attempts an error message will show.

Now, we all know that an ip address can be changed pretty quickly. A simple script can be written to brute force passwords and change the ip every 3 attempts.   
To avoid that we had to add another security layer. This is where the reCAPTCHA steps in.   
The reCAPTCHA will block the access to any web robot.  
more on reCAPTCHA in Browser Automation vulnerability.

XSS Cross-site Scripting

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.



Email, sms etc.

Reflected XSS

Reflected Cross-site Scripting (XSS) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

Reflected XSS are the most frequent type of XSS attacks found in the wild. Reflected XSS attacks are also known as non-persistent XSS attacks and, since the attack payload is delivered and executed via a single request and response, they are also referred to as first-order or type 1 XSS.

In our case, every format submission ,such as login, register and review, will redirect the user to the `submitted.php` page.

Attack scenario :

1. go to register page
2. fill out the register form
3. submit
4. in this step, you will be redirected to the submitted page.  
   for each form we would like to present a different message in the page.  
   that’s why, in our implementation, when a user is being redirected to the `submitted.php` page, we pass two parameters as arguments to this page to be printed on the page.  
     
   the redirection is implemented with the GET method, this is why the two passed parameters are seen in the url.

http://rozac.byethost7.com/submitted.php?msg=\_\_\_\_\_\_\_\_\_ \_\_\_\_\_

<h1>Logged in successfully</h1></br><a href='index.php'>go to index </a>.

1. you can clearly see that the message is actually html tags.  
   you can change it to any html code you would like.  
   the most common hack is to use javascript code inside <script> tags.  
   for example, change the message in the url   
   (everything that is shown after `msg=`) to: <script>alert('xss');</script>   
   press enter, and you will get an alert right after the page is loaded.  
   this example is a very simple one, and frankly not so dangerous. But, imagine what one could do when he has the possibility to insert any script he desires to the webpage.

Secure implementation

**<?php**// echo $\_GET['msg'];  
**echo** filter\_input(*INPUT\_GET*, "msg", *FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS*);  
  
**if** ($\_GET['target']) {  
 **echo** " you will be redirected in 5 seconds<br/>";  
 $target = filter\_input(*INPUT\_GET*, "target", *FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS*);  
 $url = 'http://' . $\_SERVER['HTTP\_HOST']; // Get the server  
 $url .= rtrim(dirname($\_SERVER['PHP\_SELF']), '/\\'); // Get the current directory  
 $url .= '/' . $target; // relative path  
 header('Refresh: 5; URL=' . $url);  
 **exit**();//important:http://thedailywtf.com/articles/WellIntentioned-Destruction  
}  
**?>**

Above code is written in the secured version. As seen before, to prevent unexpected behavior from the client side, we use *FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS* .

This filter is aware of the default charset and if a sequence of bytes is detected that makes up an invalid character in the current character set then the entire string is rejected resulting in a 0-length string.

Stored XSS

Stored XSS is the most dangerous type of Cross Site Scripting. Web applications that allow users to store data are potentially exposed to this type of attack.

Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user’s browser under the privileges of the web application.

In our case, the vulnerable website can be exploited to stored xss by pasting a script inside a review post.

How does it work:

As a respected thriving business , Roza Café strives to learn from its shortcomings and to preserve its advantages. This is the reason why we have a reviews page where you can post your review and see all reviews from other registered users.

Attack scenario:

1. register or log in with existing user
2. go to Review page
3. choose rating
4. in the review text box insert a script.  
   for example, insert <script>alert('xss');</script>
5. submit

current review is being saved in our data base and would be displayed to everyone that requests this page, thus the script would run on each and every end user’s browser.

The potential damage of the stored xss is way bigger than the persistent xss. Current code really does nothing serious but a well-crafted script could heavily affect every user that browse the review page.Secure implementation

Similarly to previous resolution, we should filter special marks (such as < , ‘ , “ , > , & , | , etc… ).

we should check that the loaded data from database is not being executed on page load.

We should never use variables directly in our queries – there are specifically designed stored quires that allow binding parameters to them and safely executing a query In our case .

Filtering clients input in review submit:

$sql = "SELECT user\_id FROM reviews WHERE user\_id = ?";

$message = filter\_input(*INPUT\_POST*, "message", *FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS*, *FILTER\_FLAG\_NO\_ENCODE\_QUOTES*);//Equivalent to calling htmlspecialchars() with ENT\_QUOTES set.

$sql = "INSERT INTO reviews (rating,content,user\_id) VALUES (?, ?, ?)";  
**if** ($stmt = mysqli\_prepare($link, $sql)) {  
 mysqli\_stmt\_bind\_param($stmt, "isi", $rating, $message, $\_SESSION["id"]);  
  
 **if** ($response->success) {  
 **echo** "<script>alert(' verified');</script>";  
 **if** (mysqli\_stmt\_execute($stmt)) {  
 // Redirect to login page  
 header("location: submitted.php?msg=<h1>Vote success</h1></br>View your and other reviews<br/>&target=reviews.php");

Similarly to previous secure implementation, we use the built-in php filter to prepare the sql query.

Browser automation with Selenium

Selenium automates browsers. That's it! What you do with that power is entirely up to you.

Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that.

Selenium can be used in variety coding languages. We used it with Python. Of course in our case, we did not use it for tests.

How does it work:

Download a browser driver (we downloaded chromedriver).

Give it a url to open (our website).

Select elements by css selectors on that page.

Manipulate these elements (change value, fill in an input fields, click a button).

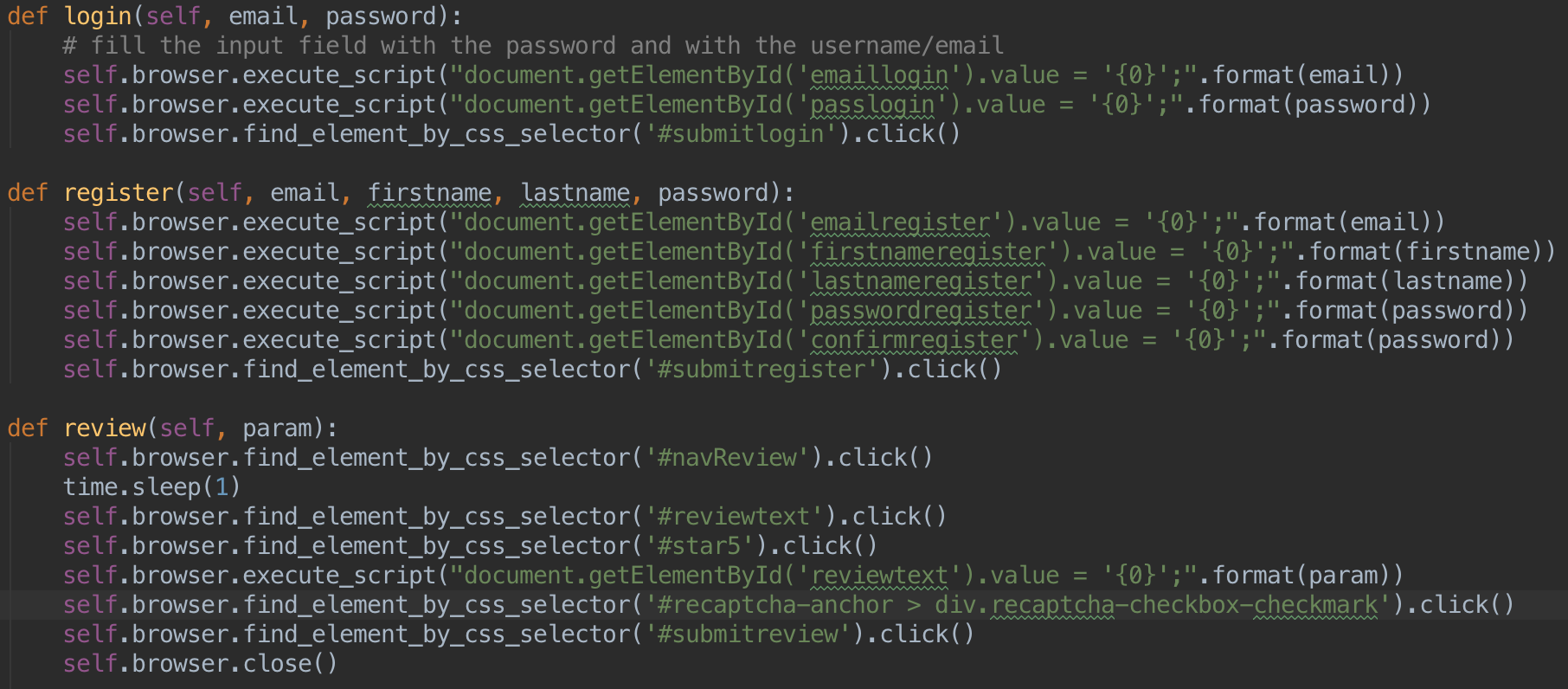
Generally, anything a user can do in a website, so can Selenium.

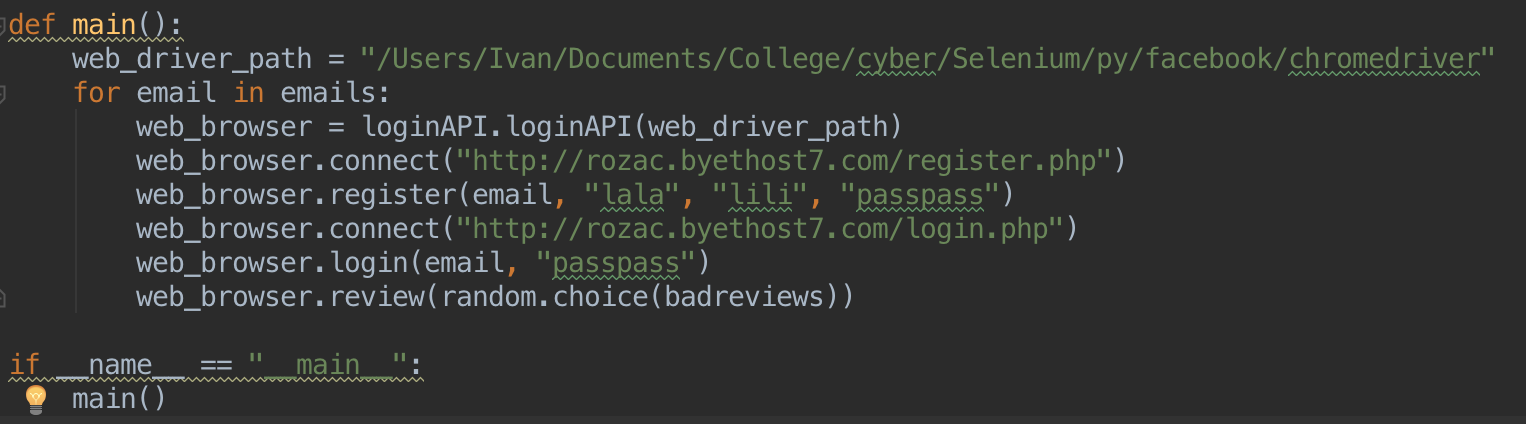
Attack scenario:

let’s assume you are an owner of a nearby café, and the new beautiful website of the Rosa Café threatens your business.  
you would then be interested in its failure.

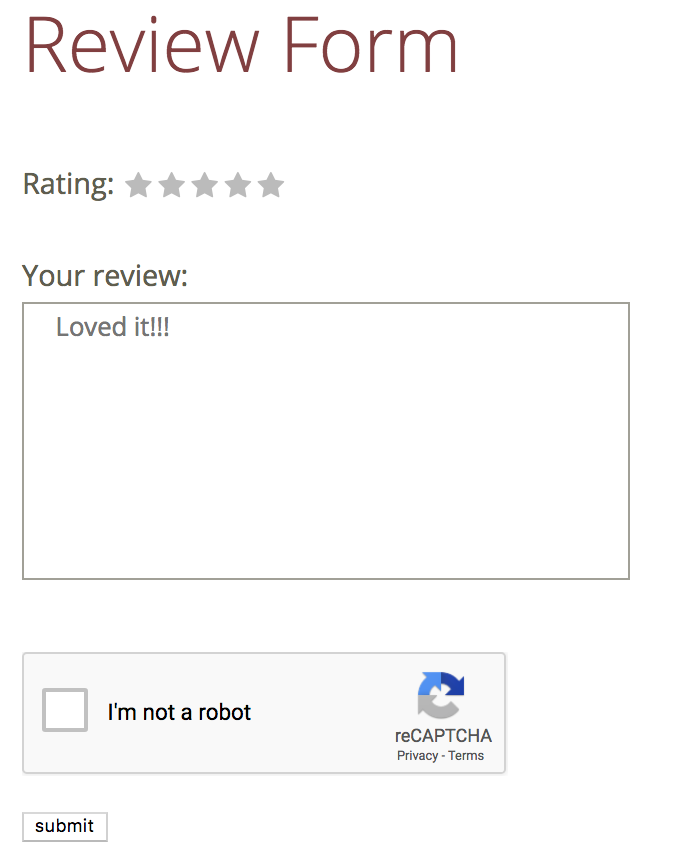
using Selenium, assuming that the website does not have a proper protection, one could, for example, open a huge amount of fictive users and submit a bad review with each of them.

1. open our website with selenium (we used chrome driver)
2. go to register page
3. register a new user
4. go to review page
5. submit a negative review
6. repeat from stage 1

A simple python script that might get our café out of business.



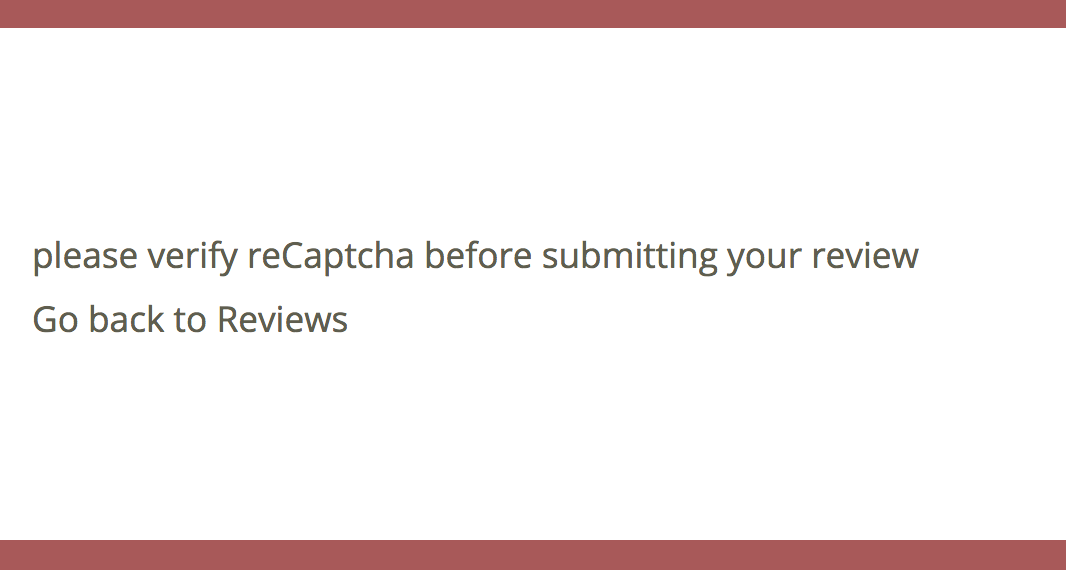
Secure implementation:

Since the Selenium is built to mimic a real user it is very challenging to identify it and overcome its attacks.  
As usual, our best friend Google has got our back!  
Google developers have created (or bought) the reCAPTCHA.  
  
reCAPTCHA is a free service that protects your website from spam and abuse. reCAPTCHA uses an advanced risk analysis engine and adaptive CAPTCHAs to keep automated software from engaging in abusive activities on your site. It does this while letting your valid users pass through with ease.  
  
reCAPTCHA offers more than just spam protection. Every time our CAPTCHAs are solved, that human effort helps digitize text, annotate images, and build machine learning datasets. This in turn helps preserve books, improve maps, and solve hard AI problems.  
  
reCAPTCHA doesn’t depend solely on text distortions to separate humans from machines. Rather it uses advanced risk analysis techniques, considering the user’s entire engagement with the CAPTCHA, and evaluates a broad range of cues that distinguish humans from bots.



In our case, when submitting a review we added a captcha which the user must verify in order to paste his review:

If a user (or a bot) has submitted the form without verifying the captcha, a proper message will be shown:



reCAPTCHA implementation:

in order to use a reCAPTCHA you must have a google account.

Once that taken care of, you can register your host in your reCAPTCHA manager and get a key for that specific host. This key can be used for that host only.

In our case we store it in $secretKey

$secretKey = '6Lfbb2sUAAAAAGMCiCUV9SBm3xcemfJQ0VD-IlMO';  
$responseKey = $\_POST['g-recaptcha-response'];  
$userIP = $\_SERVER['REMOTE\_ADDR'];  
$url = "https://www.google.com/recaptcha/api/siteverify?secret=$secretKey&response=$responseKey&remoteip=$userIP";  
$response = file\_get\_contents($url);  
$response = json\_decode($response);

**if** ($response->success) {  
 header("location: submitted.php?msg=<h1>Vote success</h1></br>View your and other reviews<br/>&target=reviews.php");  
}

$responseKey is the answer we get from reCAPTCHA once the form is submitted.

This two parameters are sent to google via URL. The response is translates to string and the to a JSON object.

If the reCAPTCHA was submitted correctly we get value true in $response->success (the generated JSON object). Only then we let the user proceed to the desired page.

Robots.txt

Web Robots (also known as Web Wanderers, Crawlers, or Spiders), are programs that traverse the Web automatically.

Search engines such as Google use them to index the web content, spammers use them to scan for email addresses, and they have many other uses.

### In a nutshell

Web site owners use the /robots.txt file to give instructions about their site to web robots;   
this is called The Robots Exclusion Protocol.

It works likes this: a robot wants to visit a Web site URL, say http://www.example.com/welcome.html. Before it does so, it firsts checks for http://www.example.com/robots.txt, and finds:

User-agent: \*

Disallow: /

The "User-agent: \*" means this section applies to all robots. The "Disallow: /" tells the robot that it should not visit any pages on the site.

In our case, robots.txt looks like this:

User-agent: \*

Disallow: /admin/userstable.php

Disallow: /admin/\*

In our case, we have a special field in the menu navigator to access this file.  
this field is enabled only if logged in with the admin user.  
The reason for that is that this page shows sensitive information about registered users and should not be seen by anyone but the admin.

Attack scenario:

1. an experienced hacker knows such file may exist and if it does, it must be at `host.domain/robots.txt`.  
   paste `/robots.txt` at the end of the site url
2. check which files the site developer doesn’t want to be checked and check them.
3. Once the attacker pastes `/admin/userstable.php` at the end of the site’s url, he gets a page that shows all data from all our registered users, including the passwords.

Secure implementation:  
  
Do not rely upon robots.txt for security!

We will show an alternative to changechange the page-level indexing settings to allow you to control how Google makes content available through search results. You can specify these by including a meta tag on (X)HTML pages or in an HTTP header.

The <meta name="robots" content="noindex" /> tag or directive applies to search engine crawlers. To block non-search crawlers, such as AdsBot-Google, you might need to add directives targeted to the specific crawler. In our case, the header of userstable.php looks like this:

<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta name="robots" content="noindex" />

The robots meta tag in the above example instructs most search engines not to show the page in search results. The value of the name attribute (robots) specifies that the directive applies to all crawlers. To address a specific crawler, replace the robots value of the name attribute with the name of the crawler that you are addressing. Specific crawlers are also known as user-agents (a crawler uses its user-agent to request a page.) Google's standard web crawler has the user-agent name Googlebot. To prevent only Googlebot from crawling your page, update the tag as follows:

<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta name="googlebot" content="noindex" />

If you need to specify multiple crawlers individually, it's okay to use multiple robots meta tags:

of course it does not disallow a user to get to a file, thus we should add a proper security to pages that contain sensitive information. Security such as reauthentication or an access condition that allows access to only specific logged in users.

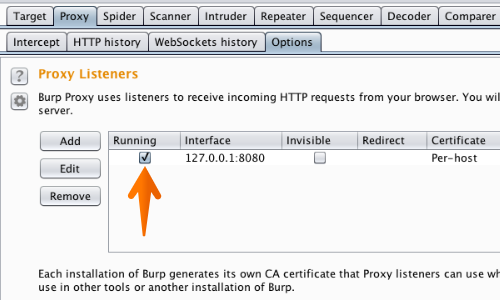
Bypassing Input Validations

# It is common to see customized client-side input validation implemented within scripts. Client-side controls of this kind are usually easy to circumvent; it is possible to enter a benign value into the input field in the browser, intercept the validated submission with your proxy, and modify the data to your desired value.

Using Burp to Bypass Client Side JavaScript Validation

### First of all, we should confirm that the proxy listener is active

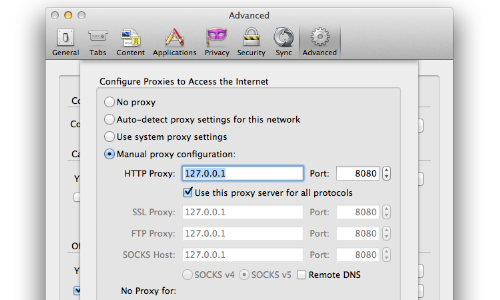
Then, we need to confirm that Burp's proxy listener is active and working on the same ip.



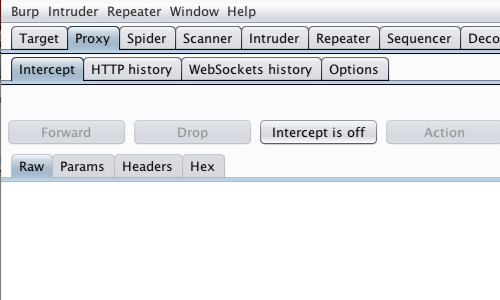
Go to the "Proxy" tab, then the "Options" sub-tab, and look in the "Proxy Listeners" section.

You should see an entry in the table with the checkbox ticked in the Running column, and "127.0.0.1:8080" showing in the Interface column. If so, please go to the section "Configure your browser to use the proxy listener" below.

### Configure our browser to use the proxy listener

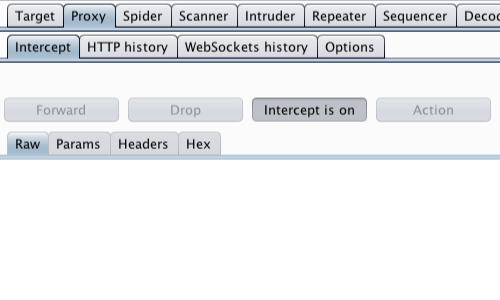


Secondly, you need to configure your browser to use the Burp Proxy listener as its HTTP proxy server. To do this, you need to change your browser's proxy settings to use the proxy host address (by default, 127.0.0.1) and port (by default, 8080) for both HTTP and HTTPS protocols, with no exceptions. The details of how to do this vary by browser and version, please use the links below to find out how to configure your browser:



With intercept turned off in the Proxy "Intercept" tab, visit the web application you are testing in your browser.

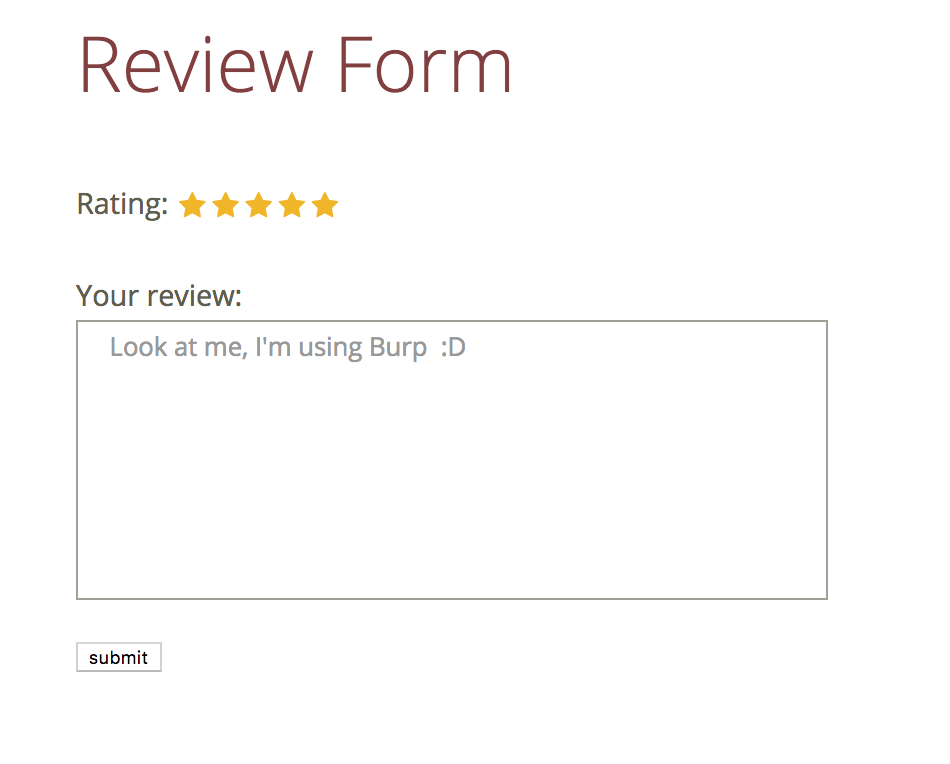
Access the page of the web application you wish to test.



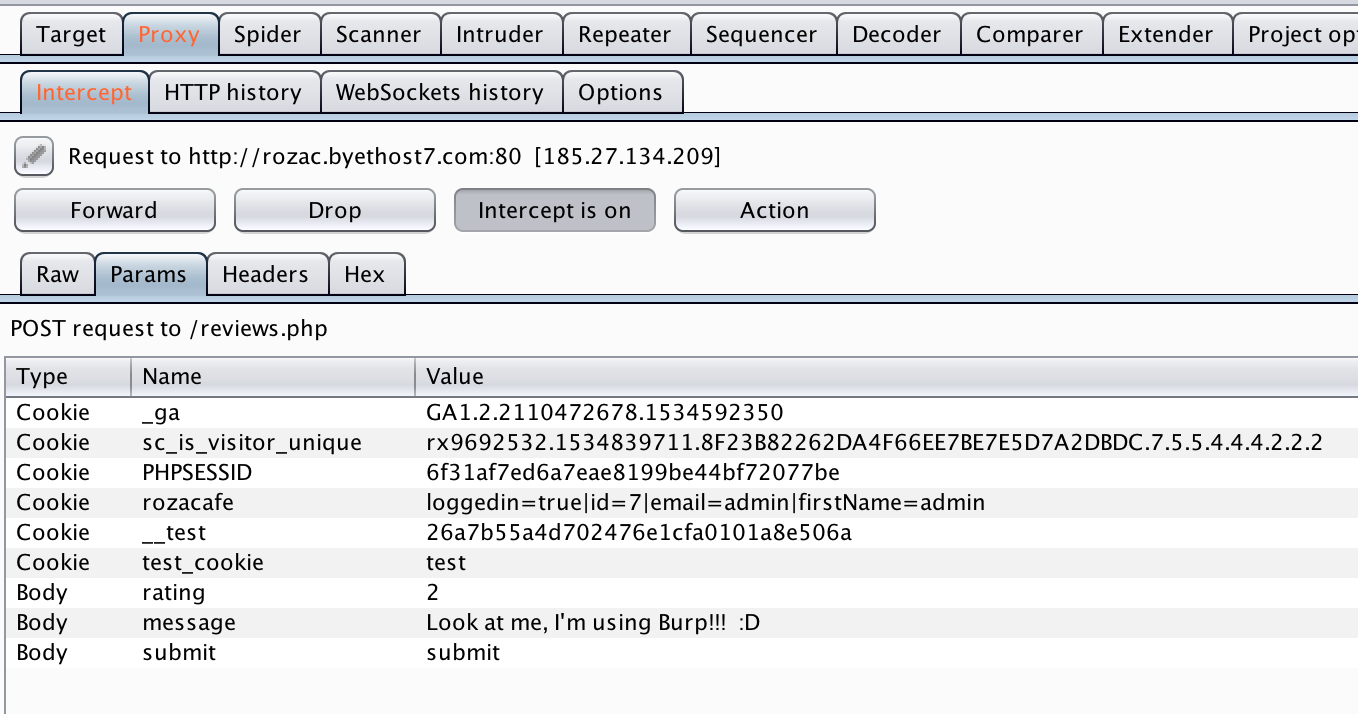
In the Proxy "Intercept" tab, ensure "Intercept is on".

Return to your browser.

Enter a benign value into the input field of your browser.

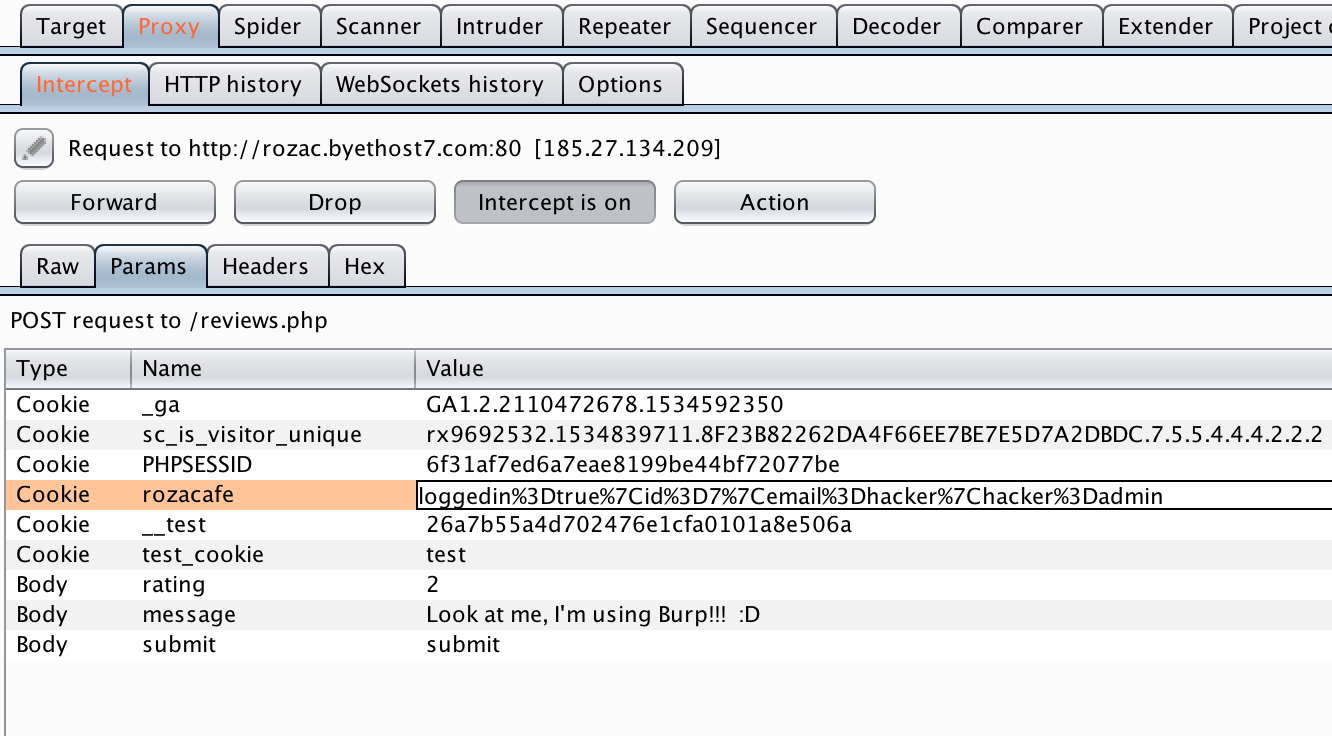


Submit the request to the server, in this example by clicking the "Submit" button.



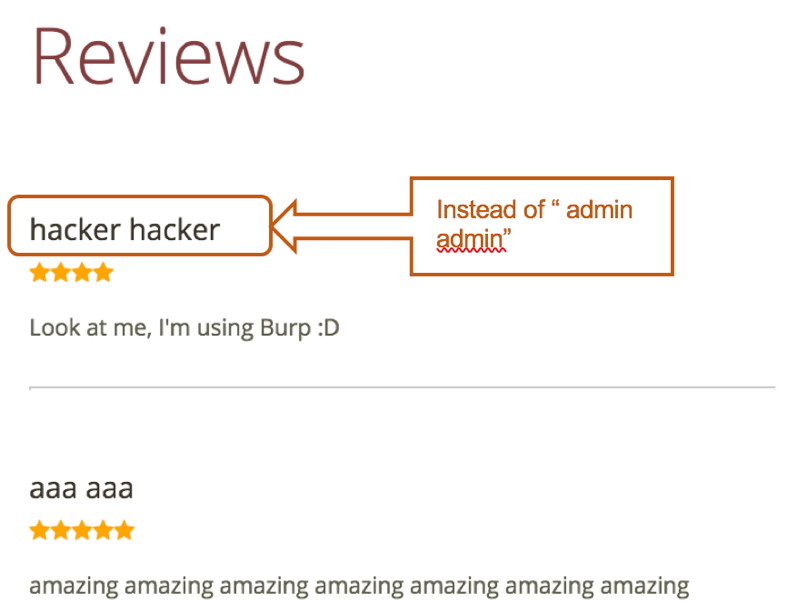
Burp will capture the request.

In this example we have used the "Params" tab to easily identify and edit the appropriate fields in the request.



Use the "Forward" button to send the request to the server.

The webpage reloads and we see the following result:



The name that should have been pasted was the name of the logged in user, but here we have been able to bypass client-side JavaScript validation.

Secure implementation:

To understand the secure version, first we will show you the code from our vulnerable version.

In our non-secure version we used a simple, non-encrypted, cookie to store different kinds of data about the currently logged in user and his actions when submitting a form.

We used a very common global built in php variable $\_COOKIE to store data in a cookie on the users machine.

$session\_arr = break\_cookie($\_COOKIE['rozacafe']);

**function** break\_cookie($cookie\_string)  
{  
 $array = explode("|", $cookie\_string);  
 **foreach** ($array **as** $i => $stuff) {  
 $stuff = explode("=", $stuff);  
 $array[$stuff[0]] = $stuff[1];  
 **unset**($array[$i]);  
 }  
 **return** $array;  
}  
  
$session\_arr = **array**();  
// get array from cookie by using the  
**if** (**isset**($\_COOKIE['rozacafe'])) {  
 $session\_arr = break\_cookie($\_COOKIE['rozacafe']);  
} **else** {  
 $session\_arr = **array**(  
 'loggedin' => 'false',  
 'id' => '0',  
 'email' => '',  
 'firstName' => ''  
 );  
}

This cookie can be read and edited in BURP.

In our secure version we use php session management system.

**if** (mysqli\_stmt\_fetch($stmt)) {  
 **if** (password\_verify($password, $hashed\_password)) {

// only secure passwords created with password\_hash will be acceppted  
  
 // Password is correct, start a new session  
 session\_start();  
 // Store data in session variables  
 $\_SESSION["loggedin"] = **true**;  
 $\_SESSION["id"] = $id;  
 $\_SESSION["email"] = $email;  
 $\_SESSION["firstName"] = $firstName;

session\_start() method initializes the session data. Return true if the session was successfully started, otherwise false. The method receives an array as an optional argument. This is an associative array of options that will override the currently set session configuration directives.

In the general situation :

* the session id is sent to the user when his session is created.
* it is stored in a cookie (called, by default, PHPSESSID)
* that cookie is sent by the browser to the **server** with each request
* the server (PHP) uses that cookie, containing the session\_id, to know which file corresponds to that user.

The data in the sessions files is the content of $\_SESSION, serialized and is un-serialized when the file is loaded by PHP, to populate the $\_SESSION array.

Sensitive information in client side

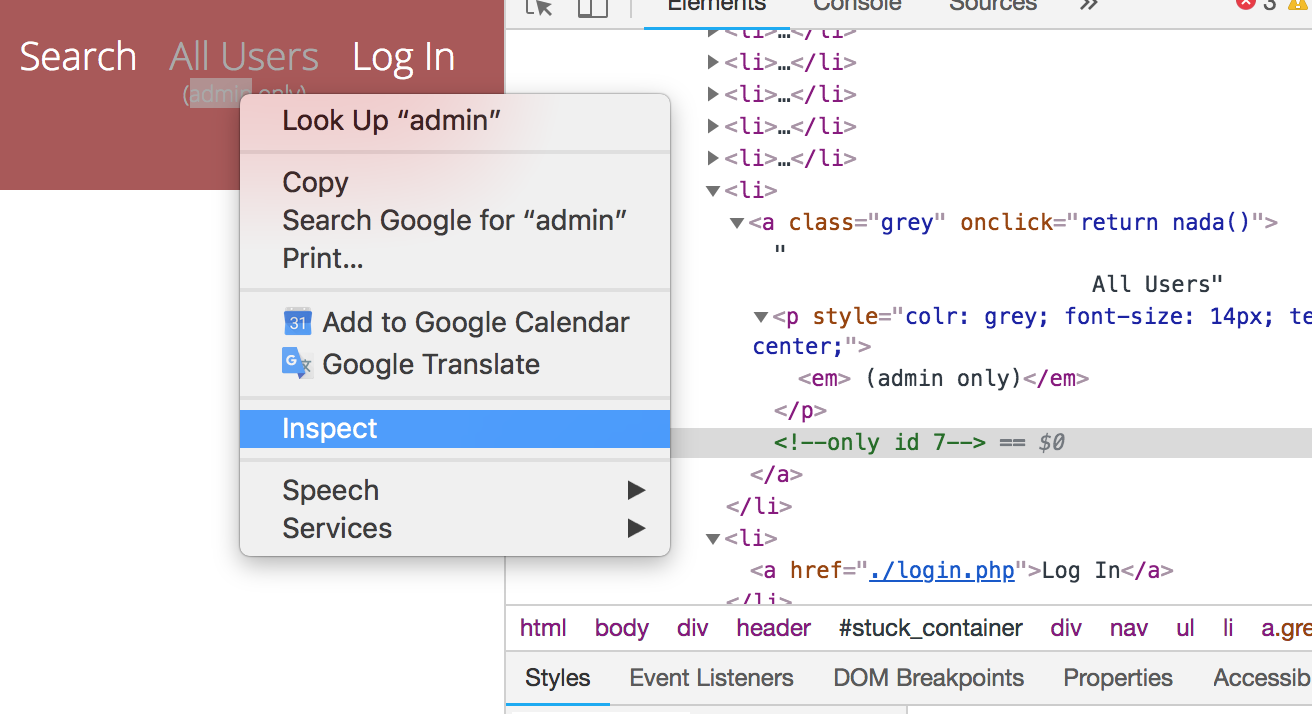
In many cases, websites allow certain actions only to authorized users.

This is made so that sensitive information will be accessible only to those who meant to see it.

While building a website the developer should check that his code works properly, especially if it involves authentications. As a developer we leave comments in the code for ourselves to explain the meaning of a scope or just to remind ourselves to do something (like the //todo comments) or it might be a comment of some sensitive information that may help us develop faster.  
  
The problem is that sometimes when we see that everything works as planned, we move on to the next challenge and forget to remove these comments.

In our case, the developer wanted to check that a certain field is allowed only for the admin user, thus he mentioned to himself near the field that it should be accessible only for a user that has the id 7.



One can inspect the DOM and see the following comment in the code.

He can now understand that 7 is the id of the admin user and can try intercept the submit processes with Burp (as we showed above) and edit the id field to 7.  
the outcome of this action might be a fake review in the best case scenario and accessing sensitive information in the worst.

Encryption

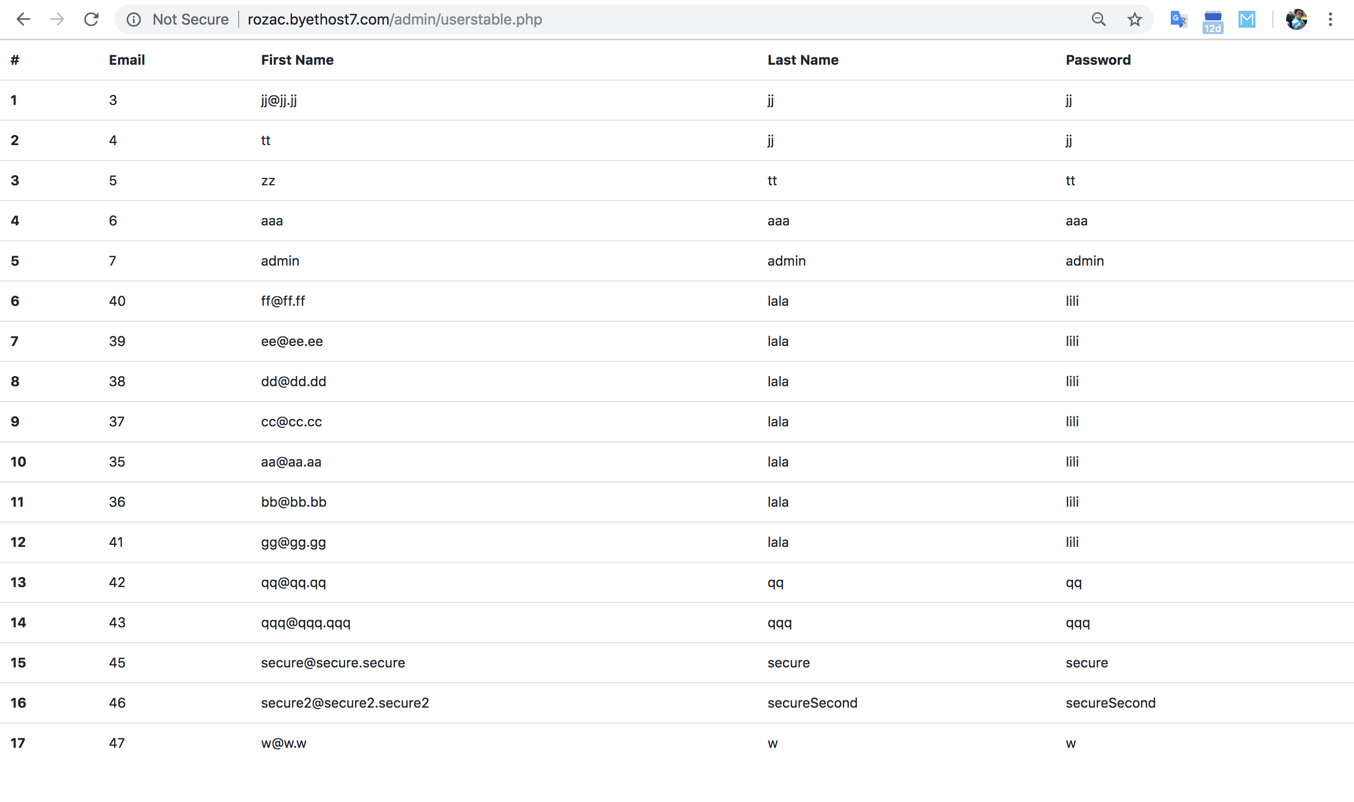
Almost all modern web applications need, in one way or another, to encrypt their users' passwords. We could say that, from the moment that an application has users, and users sign in using a password, these passwords have to be stored in an encrypted way.

There are some intuitive reasons for this: our data stores can be compromised, and so can our communications. But the most important reason is that we have to think of our users' passwords as sensitive personal data. Their passwords are their key to their privacy, so they are personal, they are sensitive, and no one (not even us) has the right to know them. And we must honor this if we want to gain our user's trust.

In our case, when a new user registers, the password is being sent without any encryption, stored in the database and as we already saw, can be seen by everyone by accessing the `userstable.php` page.

Attack scenario:

In our case, if someone enters the userstable.php he will see all the passwords of all the users:



More extensive attack can be done using SQLmap (as shown above) to get data from database.

Once a hacker gets his hand on unencrypted data he can get access to every registered user. Furthermore , many users actually use one password to many different accounts. In that case unencrypted data may be a key to hack peoples sensitive accounts (such as email, bank accounts, payment services, etc…)

Secure implementation:

In `register.php` when a user enters his password to be stored in our db we use password\_hash method built-in php since php version 5

mysqli\_stmt\_bind\_param($stmt, "ssss", $email, $firstName, $lastName, $param\_password);  
$param\_password = password\_hash($password, *PASSWORD\_DEFAULT*); // password hashing with random salt

password\_hash() creates a new password hash using a strong one-way hashing algorithm.

The method returns the hashed password, or FALSE on failure.

The following algorithms are currently supported:

PASSWORD\_DEFAULT - Use the bcrypt algorithm (default as of PHP 5.5.0). Note that this constant is designed to change over time as new and stronger algorithms are added to PHP. For that reason, the length of the result from using this identifier can change over time. Therefore, it is recommended to store the result in a database column that can expand beyond 60 characters (255 characters would be a good choice).

PASSWORD\_BCRYPT - Use the CRYPT\_BLOWFISH algorithm to create the hash. This will produce a standard crypt() compatible hash using the "$2y$" identifier. The result will always be a 60 character string, or FALSE on failure.

PASSWORD\_ARGON2I - Use the Argon2i hashing algorithm to create the hash. This algorithm is only available if PHP has been compiled with Argon2i support.

Supported options for PASSWORD\_BCRYPT:

salt (string) - to manually provide a salt to use when hashing the password. Note that this will override and prevent a salt from being automatically generated.

If omitted, a random salt will be generated by password\_hash() for each password hashed. This is the intended mode of operation.

By default, it'll use /dev/urandom to create the salt, which is based on noise from device drivers.  
  
And on Windows, it uses CryptGenRandom().  
  
Both have been around for many years, and are considered secure for cryptography (the former probably more than the latter, though).  
  
Don't try to outsmart these defaults by creating something less secure. Anything that is based on rand(), mt\_rand(), uniqid(), or variations of these is \*not\* good.

Another place we use a secure method is in our `login.php` page :

password\_verify($password, $hashed\_password)

Verifies that the given hash matches the given password.

Note that password\_hash() returns the algorithm, cost and salt as part of the returned hash. Therefore, all information that's needed to verify the hash is included in it. This allows the verify function to verify the hash without needing separate storage for the salt or algorithm information.

This function is safe against timing attacks.

Returns TRUE if the password and hash match, or FALSE otherwise.

Redirect Hijacking

//TODO Jenia