**KatCafe (**[**http://iliaal.mtacloud.co.il/**](http://iliaal.mtacloud.co.il/)**)**

Web application penetration testing assessment



**Chen Ben Shalom**, 204594626

**Ilia Altmark**, 312927080

**Dor Tasa**, 311596746

**System Description**

"KatCafe" is a platform that lets users order a table or a few tables at the "KatCafe" café and in addition to that, provides information about the cats that currently reside at the establishment.

**Table of Contents**

[Attacks Findings [1] – Broken Authentication 4](#_Toc44766385)

[Attacks Findings [2] – Broken Access Control 6](#_Toc44766386)

[Attacks Findings [3] – Reflected XSS 8](#_Toc44766387)

[Attacks Findings [4] – Stored XSS 11](#_Toc44766388)

[Attacks Findings [5] – LFI 13](#_Toc44766389)

[Attacks Findings [6] – RFI 15](#_Toc44766390)

[Attacks Findings [7] – Phishing Redirect 18](#_Toc44766391)

[Attacks Findings [8] – SQL Injection 21](#_Toc44766392)

[Attacks Findings [9] – Command Injection 24](#_Toc44766393)

[Attacks Findings [10] – Malicious File Upload 26](#_Toc44766394)

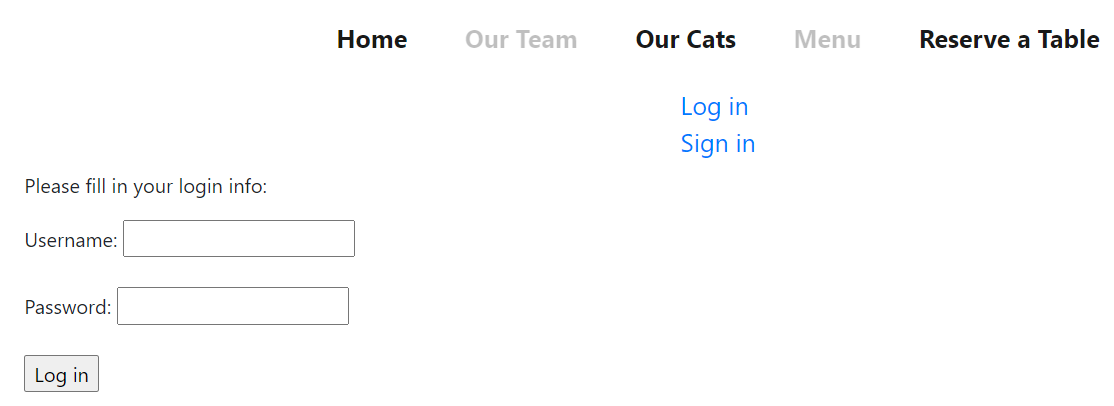
# Attacks Findings [1] – Broken Authentication

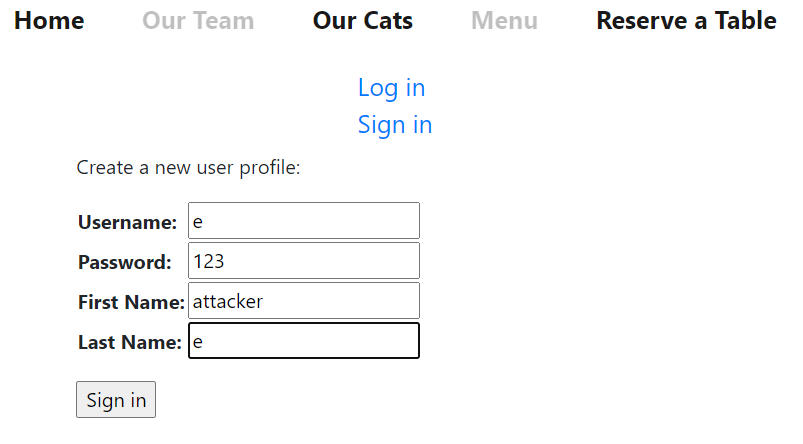
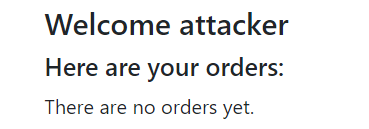
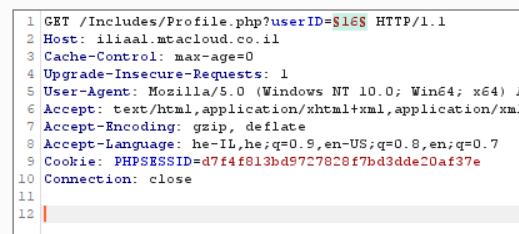
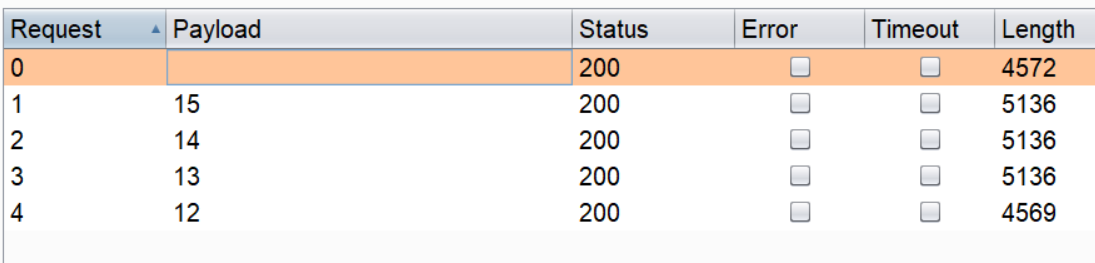
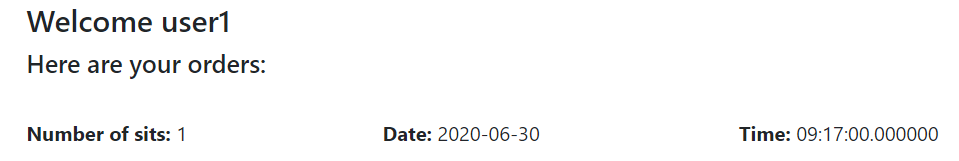
1. **Broken authentication**
   1. **Description**

The attacker can get access to the profile pages of all the other users without using proper authentication. The attacker simply types in the user id in the URL when accessing the profile page.

* 1. **POC**
     1. The attacker enters the site and tries to access the Profile page. He gets redirected to the login page.

<http://iliaal.mtacloud.co.il/Includes/login.php>

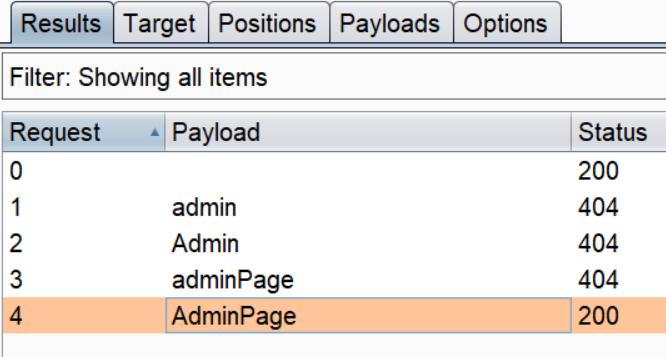
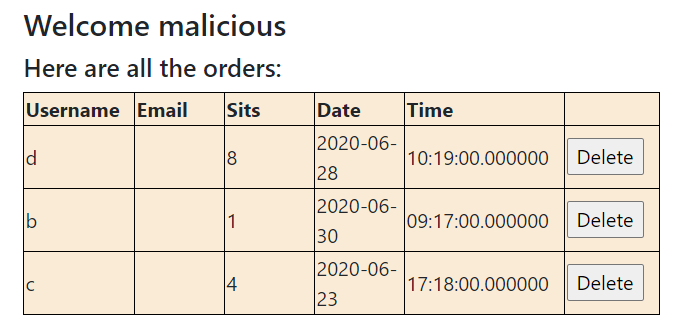


* + 1. The attacker signs in with a new account. 
    2. After signing in the attacker enters the profile page. He can see the following URL: [http://iliaal.mtacloud.co.il/Includes/Profile.php?userID=16](http://iliaal.mtacloud.co.il/Includes/Profile.php?userID=16)
    3. The next step is to try and access other user profile pages by changing the userID value which is sent to the server. We will use intruder in burp: 
    4. The attacker (The reason why he is the attacker and not a malicious user is because he does not have to be logged in to perform the attack. The attacker created an account to find the vulnerability) is using a list to guess the user id's: 
    5. As we can see the attacker can get access to other user profile pages and their order info. [http://iliaal.mtacloud.co.il/Includes/Profile.php?userID=13](http://iliaal.mtacloud.co.il/Includes/Profile.php?userID=13)
  1. **Business impact**
* The attacker can get access to personal information of other users.
* In some cases, the attacker could change login info and use the account for his own needs.
  1. **Mitigations**
* Should not send user info in the URL using GET.
* Better to rely on sessions and store and retrieve user info on the server side.

# Attacks Findings [2] – Broken Access Control

1. **Broken access control**
   1. **Description**

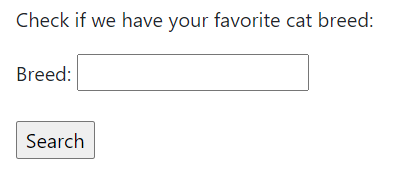
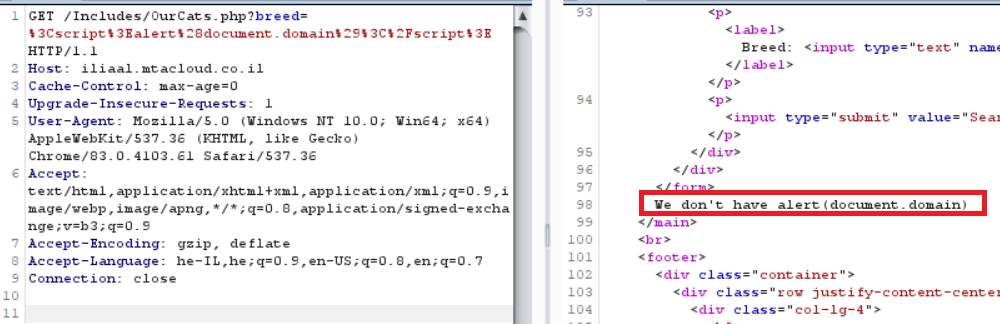
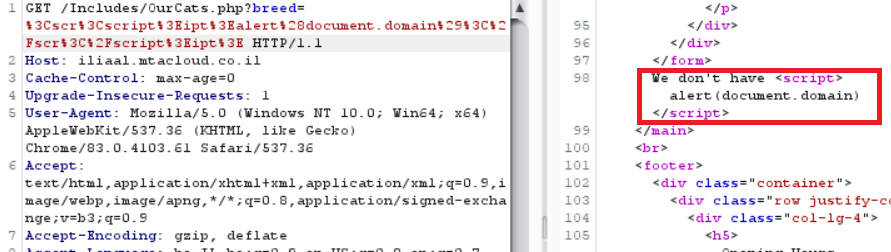
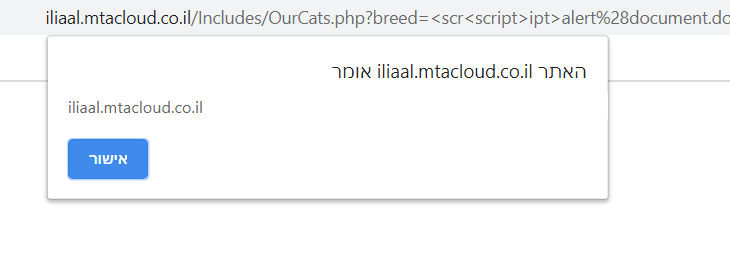
A malicious user could try and get access to admin's page in order to view, change or delete other user info. After logging in the user might find the admin page and get access even though he's not supposed to.

* 1. **POC**
     1. After logging in the user might want to look for the admin page. After looking around he might find out that all the pages other than index.php are in the folder "Includes". Considering that the admin page might be in that folder as well, the user might try to use brute force and guess the name of the page.****
     2. The malicious user eventually finds the page. <http://iliaal.mtacloud.co.il/Includes/AdminPage.php> ****
     3. The malicious user enters the admin page as a normal user (normally the admin will be redirected to the admin page after logging in. also if the user wasn't logged in he would be redirected to login page). As we can see the user can delete orders of other users even though he is not an admin.
  2. **Business impact**
* Getting access to unauthorized info and permissions to change it.
  1. **Mitigations**
* Performing server-side privilege validation before giving access to components and sensitive info.

# Attacks Findings [3] – Reflected XSS

1. **Reflected XSS**
   1. **Description**

On one of the pages of the web application (OurCats.php) there is a search bar which enables the attacker to perform RXSS. This makes the web app vulnerable to RXSS. The attacker can inject malicious JS code to the parameter (search bar value) in the URL. This code will be executed once the URL page is loaded. The attacker can use phishing methods to lure victims to access the link.

* 1. **POC**
     1. The attacker enters the page with the search bar. [http://iliaal.mtacloud.co.il/Includes/OurCats.php](http://iliaal.mtacloud.co.il/Includes/OurCats.php)
     2. The attacker injects the following JS code into 'breed' parameter in the URL. <http://iliaal.mtacloud.co.il/Includes/OurCats.php?breed=%3Cscript%3Ealert%28document.domain%29%3C%2Fscript%3E>
     3. We can see that the script tags are filtered out. 
     4. Now that we know that the script tags are filtered, we will change the code.
     5. The malicious code has been successfully injected into the HTML code. <http://iliaal.mtacloud.co.il/Includes/OurCats.php?breed=%3Cscr%3Cscript%3Eipt%3Ealert%28document.domain%29%3C%2Fscr%3C%2Fscript%3Eipt%3E> 
     6. The result is a JS code execution.
  2. **Business impact**

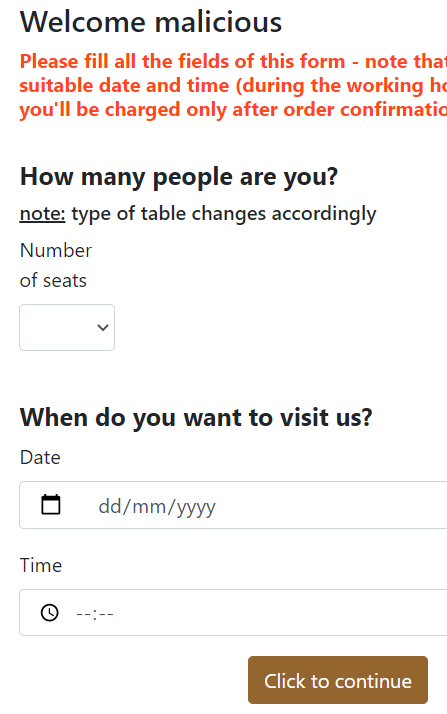
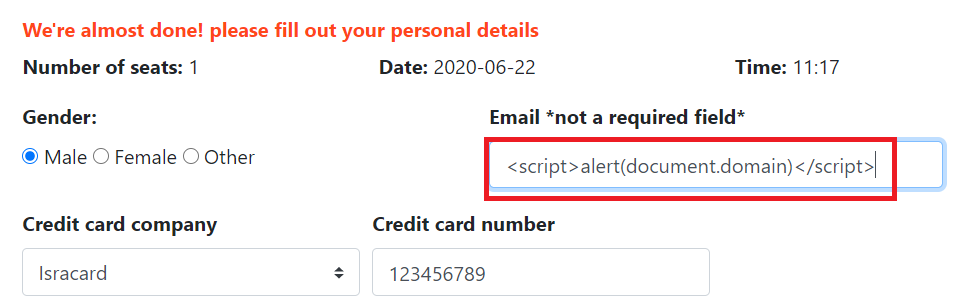
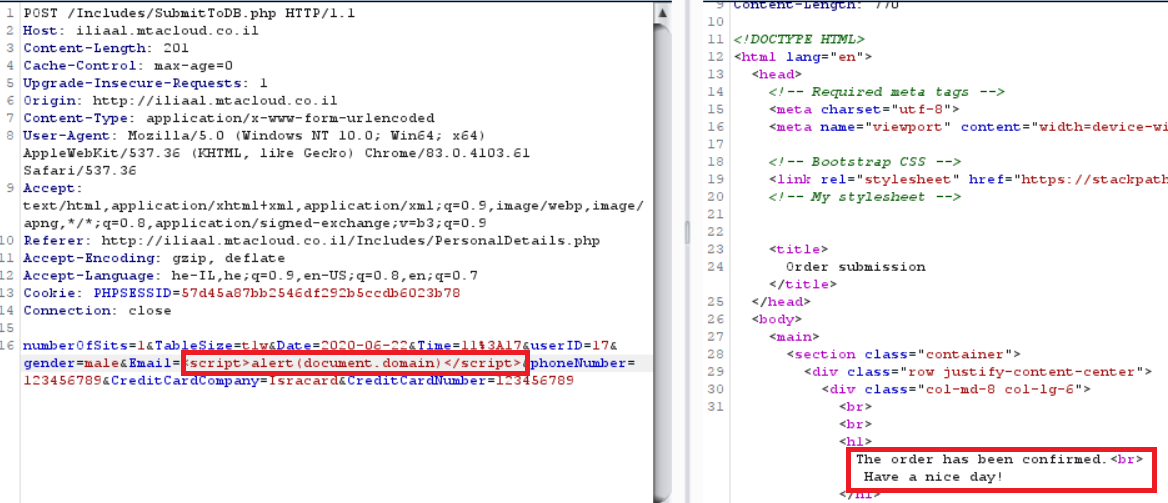
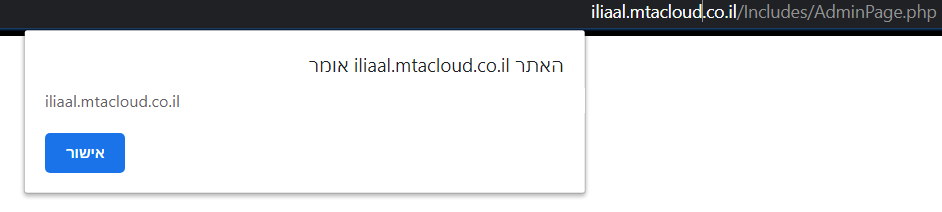
The attacker can use social engineering techniques like phishing in order to lure the victim to inadvertently make a request to the server, which contains the XSS payload and ends-up executing the script that gets reflected and executed inside the browser. The result may be:

* Victims sharing their username and password after being redirected to a fake fishing site and typing in their info into a fake login form.
* The script sending user's cookie to the attacker.
  1. **Mitigations**
* Input validation - At the point where user input is received, filter as strictly as possible based on what is expected or valid.
* Output sanitization - At the point where user-controllable data is shown in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
* Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the *Content-Type* and *X-Content-Type-Options* headers to ensure that browsers interpret the responses in the way you intend.
* Use HTTPOnly cookie flag on cookies that are not supposed to be accessed by java script.

# Attacks Findings [4] – Stored XSS

1. **Stored XSS**
   1. **Description**

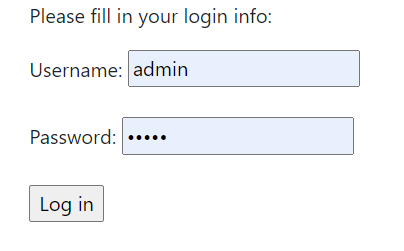
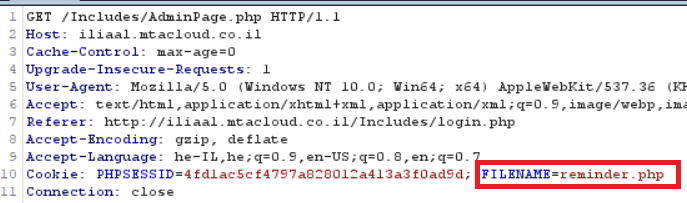
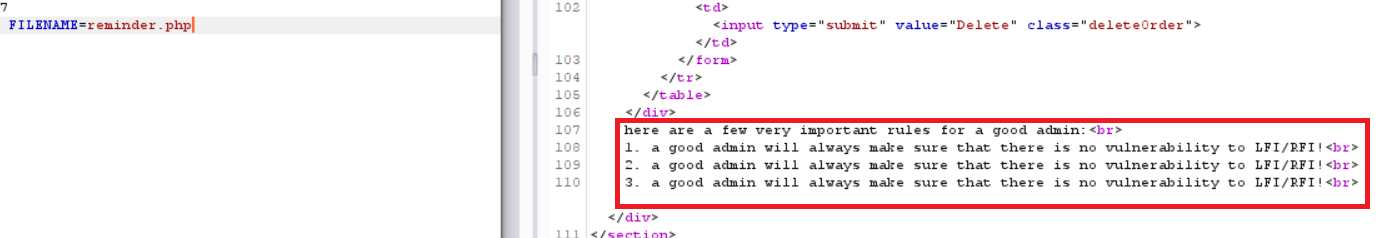
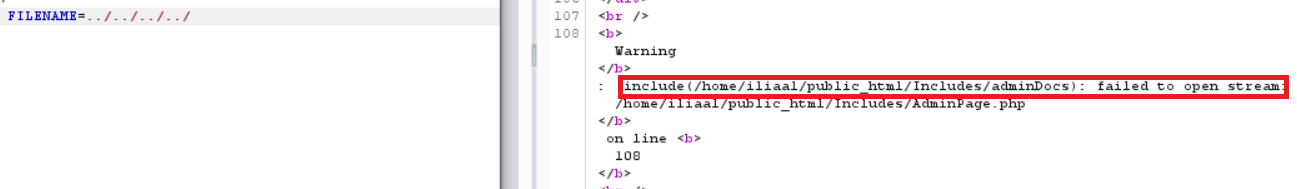
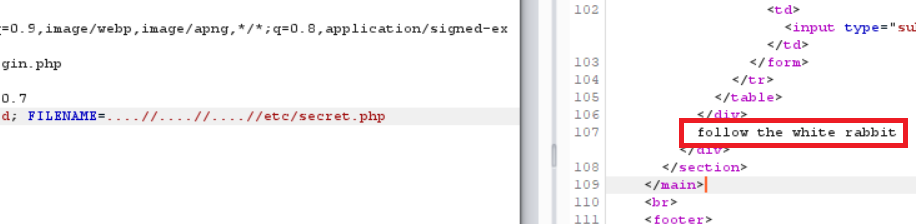
After logging in, the malicious user can inject JS code and store it in the server's database. When ordering a table the user can provide an email address. Instead of providing a valid email address the user can inject a payload which will be stored in the database.

* 1. **POC**
     1. After logging in the malicious user will click on "Reserve a Table" and provide the order details. [http://iliaal.mtacloud.co.il/Includes/ReserveATable.php](http://iliaal.mtacloud.co.il/Includes/ReserveATable.php)
     2. After filling the form, the malicious user will be redirected to another form. In that form there is an email field, the email field in the database can hold up to 70 characters meaning the user can type in a full payload and potentially cause a lot of harm.
     3. [http://iliaal.mtacloud.co.il/Includes/PersonalDetails.php](http://iliaal.mtacloud.co.il/Includes/PersonalDetails.php)
     4. Here we can see this in burp. 
     5. After the user types in the payload and sends the form, the payload is saved on the server. In order activate the payload the admin needs to enter admin page.As we can see when the admin enters admin page the script is activated.<http://iliaal.mtacloud.co.il/Includes/AdminPage.php> 
  2. **Business impact**
* Depending on the location of the payload in the website could cause a lot of harm and would activate for everyone who opens the page in their browser.
* In terms of possible damage, the impact is the same as reflected XSS while one of the more severe attacks involves stealing the user's session cookie.
  1. **Mitigations**
* Same rules apply as in reflected XSS.

# Attacks Findings [5] – LFI

1. **LFI**
   1. **Description**

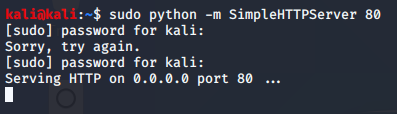
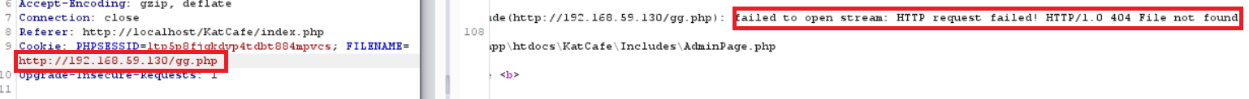
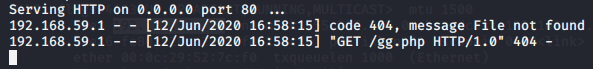
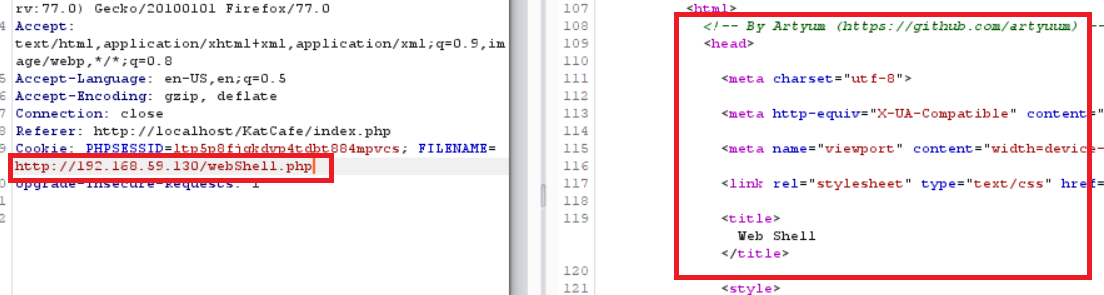
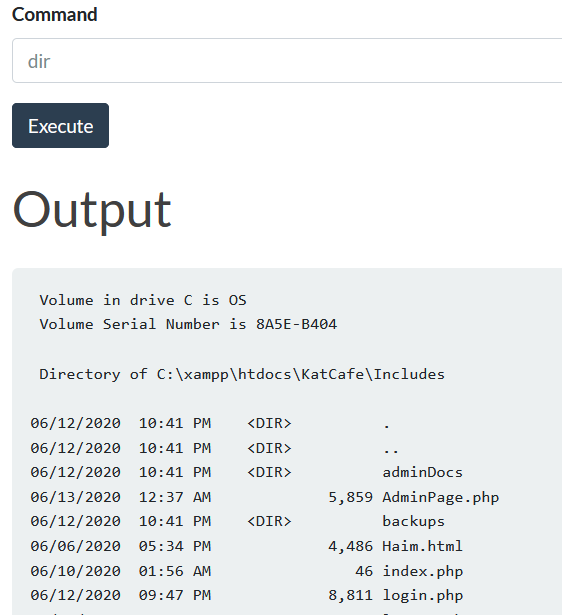
After logging in, the malicious admin can get access and execute sensitive files using the LFI vulnerability. The admin page has the include() function which draws the name of the included file from a cookie, which is created after the admin logs in.

* 1. **POC**
     1. Themalicious admin logs in. 
     2. After that, a cookie called "FILENAME" is created which holds the name of the included file. 
     3. While being redirected to AdminPage.php, the malicious admin decides to try and load a different file. This is the normal message: 
     4. Changing the FILENAME value does not seem to get us out of the site's domain (even though it should have). Perhaps "../" is being filtered out from the cookie.
     5. Taking into account the possibility that "../" is being filtered out, eventually the malicious admin finds the secrets he craves. 
  2. **Business impact**
* Hackers getting access to restricted files beyond the site's domain.
* Executing files that a normal user is not supposed to**.**
* Could potentially execute files that let the hacker run various actions (i.e. using php) including run shell commands which can potentially give the hacker total control over the server.
  1. **Mitigations**
* Should avoid using user inputs, as well as passing the info with GET or cookiesto include files.
* Validating user input.
* Using whitelist approach.

# Attacks Findings [6] – RFI

1. **RFI**
   1. **Description**

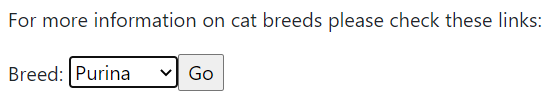
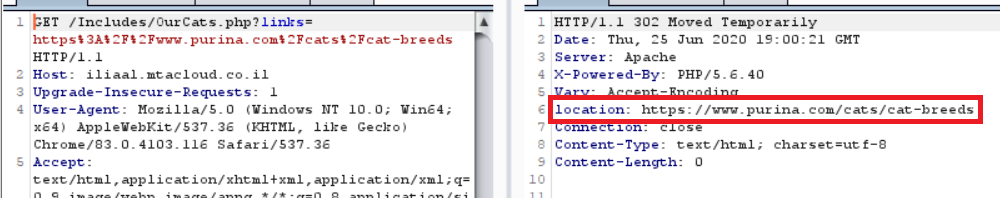
After logging in, the malicious admin can execute files from external sources (i.e. other sites and servers) using the RFI vulnerability. The admin page has the include() function which draws the name of the included file from a cookie, which is created after the admin logs in.

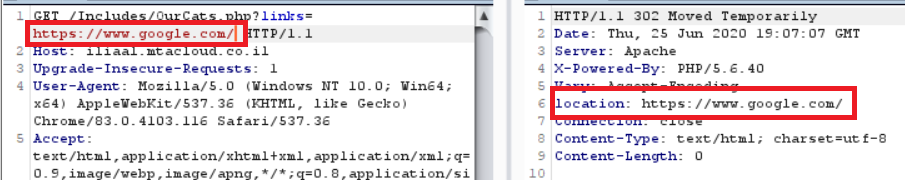
* 1. **POC**
     1. The malicious admin starts the attack with same steps as in **5.2.1** - **5.2.3** (we have changed the code a little bit for this attack. We hadto remove the full path of the file to make the web app vulnerable).
     2. The next step would be to check if the site is vulnerable to RFI. The malicious admin runs a server on his computer and is waiting for requests. ****
     3. The malicious admin checks if there is a vulnerability. And it turns out there is. We requested to open a nonexistent file and we received an error.****
     4. The next step is to prepare the web shell. 
     5. Now that the web shell is on the attacker's server, we can execute it using the RFI vulnerability.
     6. This is how it looks in a browser after running dir. 
  2. **Business impact**
* Could potentially execute files that let the hacker run various actions (i.e. using php) including run shell commands which can potentially give the hacker total control over the server.
* This can lead to shutting down the server/site or defacement.
  1. **Mitigations**
* The mitigations from the previous attack (5.4) also apply here.
* A more specific mitigations when using php can include:
  1. The php.ini file should have the parameter "allow\_url\_include" set to Off.
  2. In the same file we can disable certain dangerous functions (system, passthru, shell\_exec) in the parameter "disable\_functions".

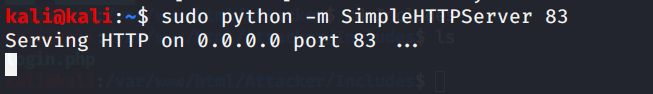
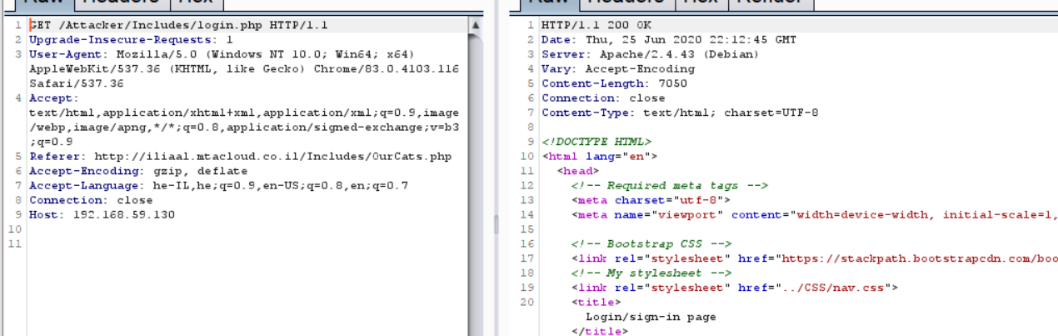
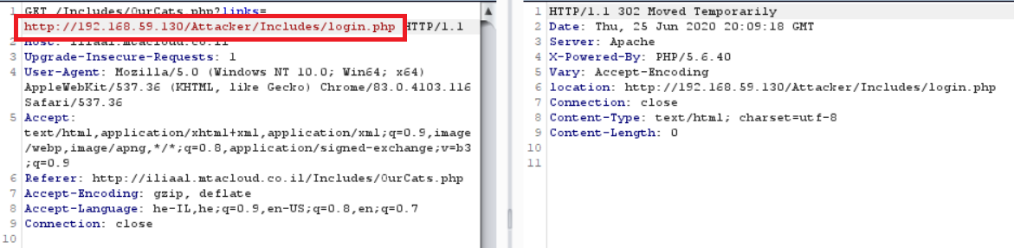
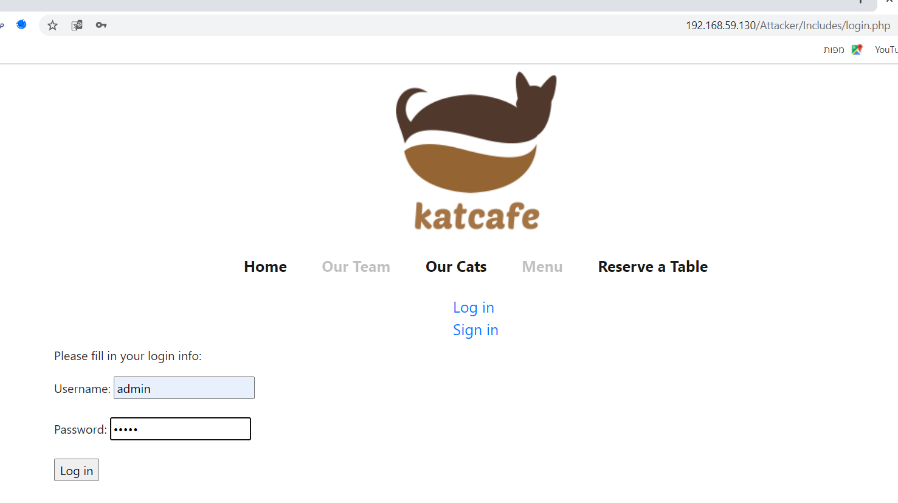
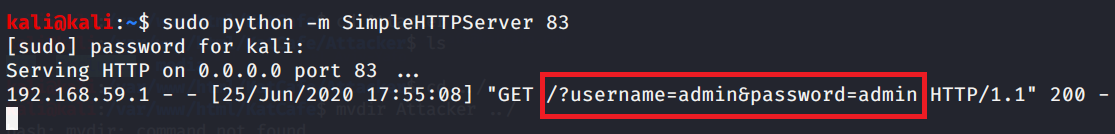
# Attacks Findings [7] – Phishing Redirect

1. **Phishing redirect**
   1. **Description**

The attacker can create a link with the site's domain using the open redirect vulnerability that redirects to a phishing site. The phishing site is used to steal user credentials where the user types in his username and password thinking it is the original site. The vulnerability is on the "Our Cats" page.

* 1. **POC**
     1. The attacker enters the "Our Cats" page and sees the select form. <http://iliaal.mtacloud.co.il/Includes/OurCats.php> ****
     2. The attacker opens Burp and sees that the site redirects to different page. A quick check and it is clear we got an open redirect vulnerability.

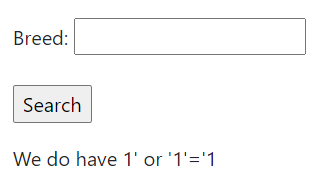
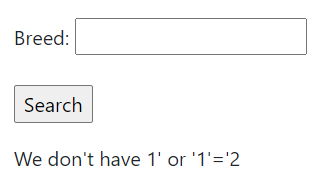
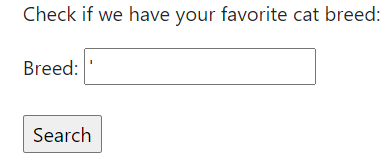
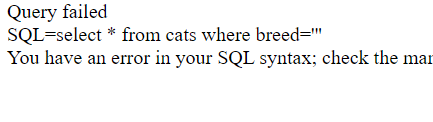
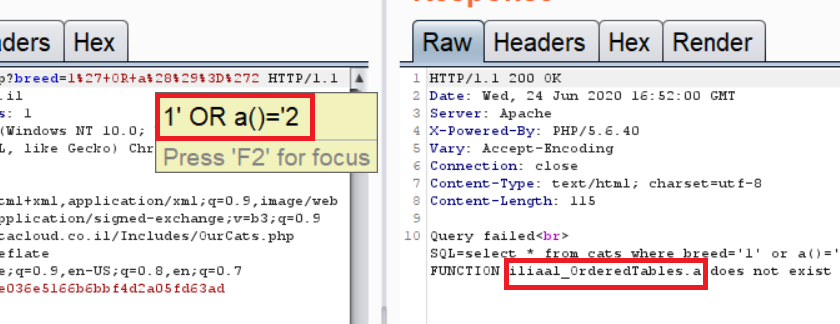
****

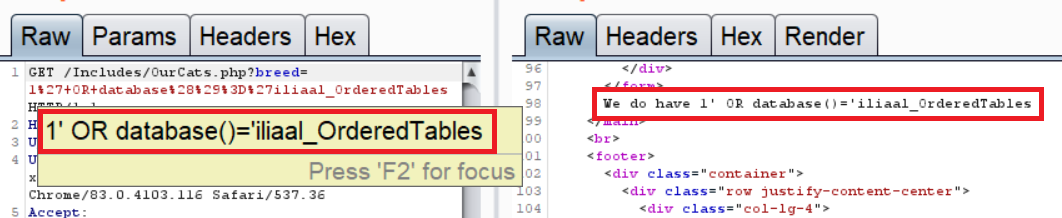
* + 1. The attacker copies the login page of the site and puts it into his server and starts the server.  ****
    2. The login page got a script added to it which will send the user's login credentials back to the attacker's server. 
    3. The attacker is also listening.
    4. The attacker is preparing to send the link to his victim. And the site looks identical. 
    5. The victim clicks on the link and enters the login credentials.
    6. The attacker receives the login credentials. 
  1. **Business impact**
* Attackers using the site’s domain to redirect to a phishing site from where user’s information is being stolen.
* The phishing site looks identical to the original, and for that reason the victim does not suspect a thing.
* The attacker’s site can have various content which can also cause XSS attacks.
  1. **Mitigations**
* Can be solved using white list approach. Redirect only to permitted sites.
* The easiest solution is to no let the user control where the page redirects to. Use a list of links instead.
* Instead of sending a link in GET (or other untrusted input) could send an ID and the appropriate link would be determined server-side.
* Check that the URL begins with *http://* or *https://* and invalidate all other URLs.

# Attacks Findings [8] – SQL Injection

1. **SQLi**
   1. **Description**

The attacker can perform error-based (/blind) SQL injection attack from the "Our Cats" page. In the bottom of the page there is an input field that the attacker can use to his advantage. By entering a certain SQL commands the attacker can find the name of the database.

* 1. **POC**
     1. The attacker enters the "Our Cats"page and tries to check for vulnerability. First, he checks how it responds to true and false statements. To true statements it responds with "We do have".  To false statements it responds with "We don't have". 
     2. Next step is checking for errors. The attacker receives an error with the query. ****
     3. The attacker enters the payload and receives something interesting. The payload has an undefined function and we receive an error with possibly the database name. 
     4. The attacker wants to confirm that it is indeed the database name. And it is confirmed, we know from previous tests that '1' would not return true which means that the statement "database()='iliaal\_OrderedTables'" is true (If the errors would not have been shown the attacker could also perform blind SQLi. The attacker could have used "1' or substring(database(),1,1)='i" in order to guess the first letter of the DB and using this method eventually guess the name of the DB).

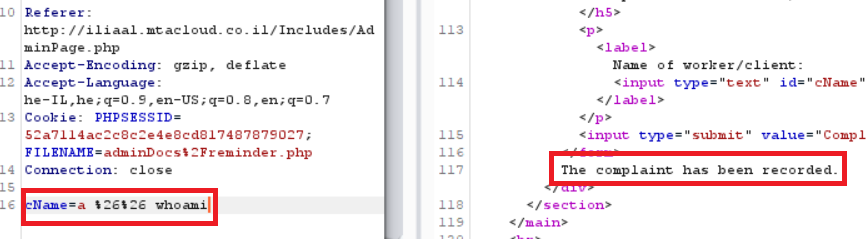
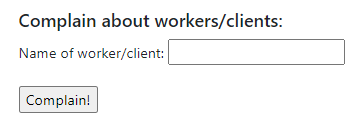
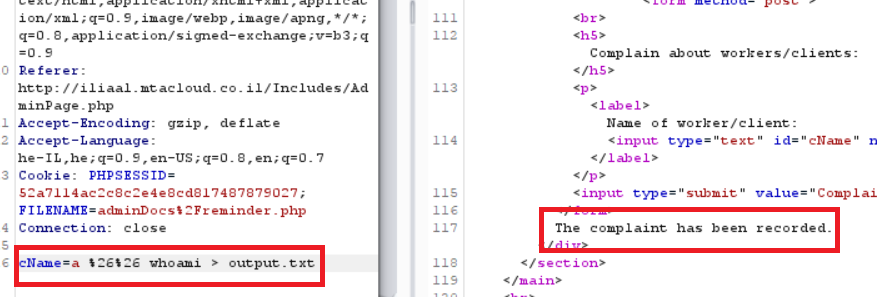
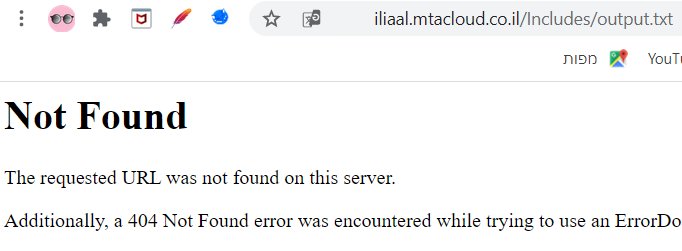
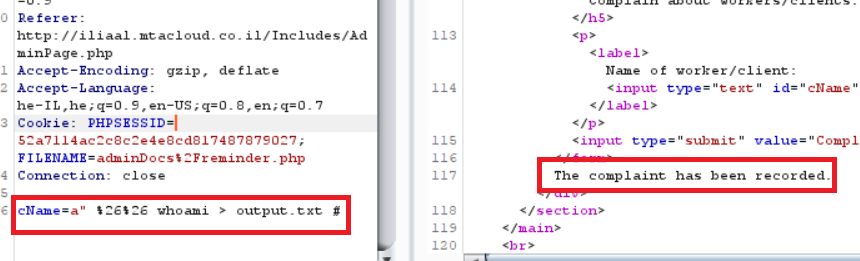
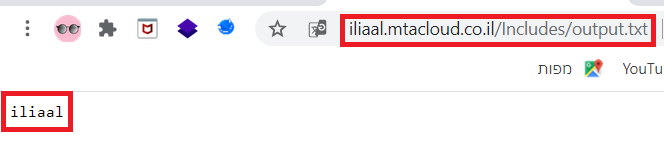


* 1. **Business impact**
* Attackers bypassing authentication and stealing user info or impersonating other users.
* Finding and exposing sensitive info on the DB.
* Damaging the integrity of stored info (by deleting, inserting and changing content).
* Gaining control over the system.
* Overloading the system with resource consuming operations to deny services from legitimate users(DDoS)
* Damaging firm reputation and the firm getting regulatory fines.
  1. **Mitigations**
* Using the defenses described in the [OWASP SQLi cheat sheet](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html), such as prepared statements, stored procedures, white-list input validation and escaping.
* Hiding the system/sql errors from users.

# Attacks Findings [9] – Command Injection

1. **Command injection**
   1. **Description**

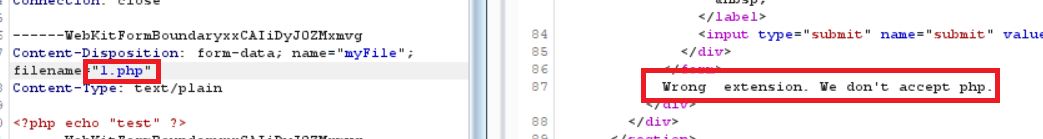
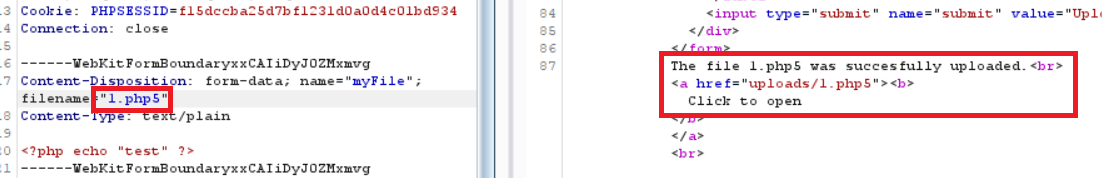
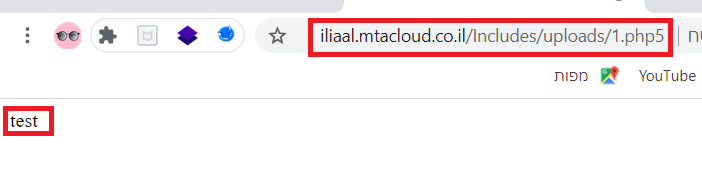
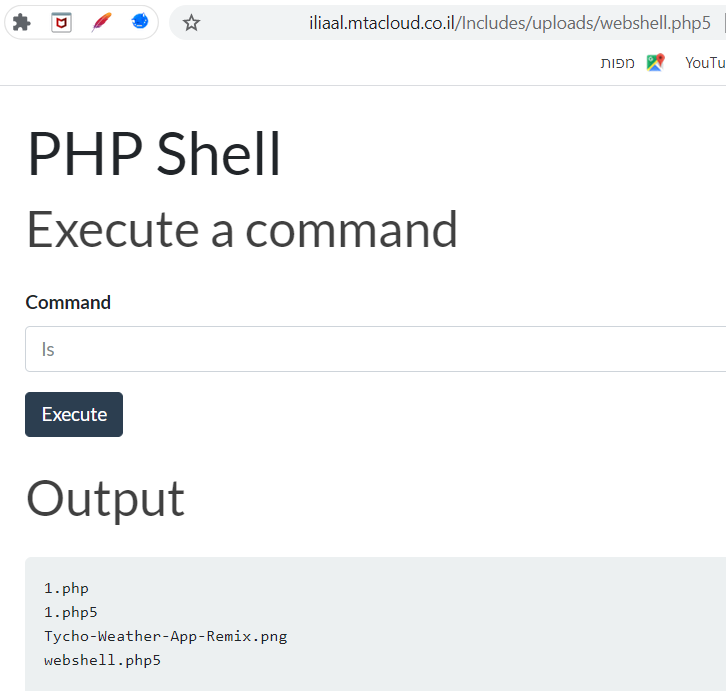
The malicious admin can inject various terminal commands which can cause damage to the server. The admin page has a form where the admin can complain about workers/clients. The input of that form is attached to a shell command. The output of that command is recorded on the server.

* 1. **POC**
     1. The malicious admin logs in and enters the admin page. On that page he tries to enter the whoami command. But the output of the command is not shown. [http://iliaal.mtacloud.co.il/Includes/AdminPage.php ](http://iliaal.mtacloud.co.il/Includes/AdminPage.php)
     2. The malicious admin decides to save the output of the command in a file.  But the file has not been created. ****
     3. The malicious admin considers the fact that the form input is part of a string and makes the next changes. 
     4. The malicious admin checks if the file has been created and indeed it worked. 
  2. **Business impact**
* The attacker can fully compromise the server's data.
* After compromising the server, the attacker can leverage the attack to compromise other servers in the organization.
  1. **Mitigations**
* Should never let a user execute system commands.
* Using white list validation of only the permitted values.
* The input should contain only alphanumeric characters and no special characters or whitespace.

# Attacks Findings [10] – Malicious File Upload

1. **Malicious file upload**
   1. **Description**

A malicious user can exploit the unrestricted file upload vulnerability to upload malicious files to the server. The profile page contains an option to upload files into a folder in the server. After the upload, the files become accessible to the malicious user, and then he executes them.

* 1. **POC**
     1. The malicious user logs inand enters his profile page. In that page he can upload files. [http://iliaal.mtacloud.co.il/Includes/Profile.php?userID=17](http://iliaal.mtacloud.co.il/Includes/Profile.php?userID=17)
     2. The malicious user decides to see what files he can upload. It looks like the server does not accept php files.
     3. The malicious user decides to use an alternative php extension. It looks like it works.
     4. The malicious user can also execute the file in the browser. 
     5. Next step is to create and upload the web shell.
     6. The malicious user executes the web shell using the browser.
  2. **Business impact**
* Attackers uploading malicious files to the server and executing them or letting others execute them.
* Attackers can gain control of the server by uploading shells and executing them, thus being able to send commands to the terminal.
* Users can open malicious files that contain viruses or some other malware.
  1. **Mitigations**
* Using white list filter to accept only the extensions that are necessary.
* The application must verify that the contents of the uploaded files do not contain malicious code.
* Some special characters should be discarded (or just do not upload) from the file name before upload.