

Navigate CMS



I recently took part in a live bug hunting event:

OnSecurity

@WeAreOnSec... · [Follow](#)



Who's excited for [@SecGus](#) going live in 1 hour with [@CalumBoal](#) for the first ever [#HackStream](#)!

Live [#BugHunting](#) on [#Stream](#) trying to find some 0 days @ twitch.tv/SecurityGus

[#TT0dayHuntingNavigate](#)

As part of these event, we reviewed the product [Navigate CMS](#). In order to perform this review I setup a local instance of this tool, using version v2.9 r1433 (2020/06).

While taking part of this event, I managed to help identify several findings:

- User enumeration
- Session information leakage
- User password reset vulnerability
- Store Cross-Site Scripting (XSS)
- Reflected Cross-Site Scripting (XSS)

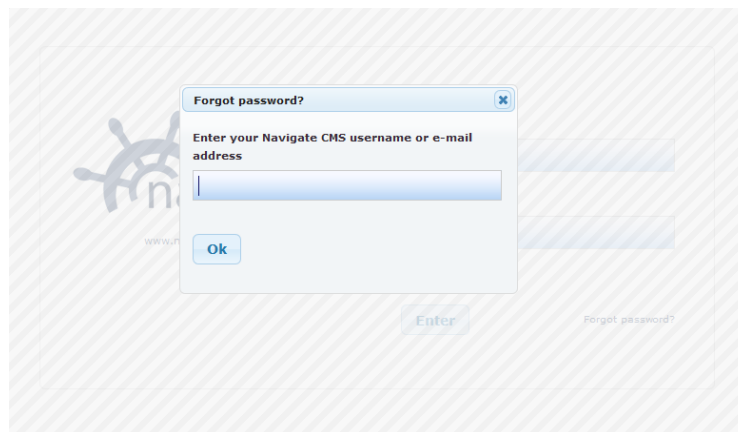
User Enumeration - CVE-2020-14016

Navigate CMS has a *Forgot password?* feature, which allows a user to reset their password if they forgot what their current password is.



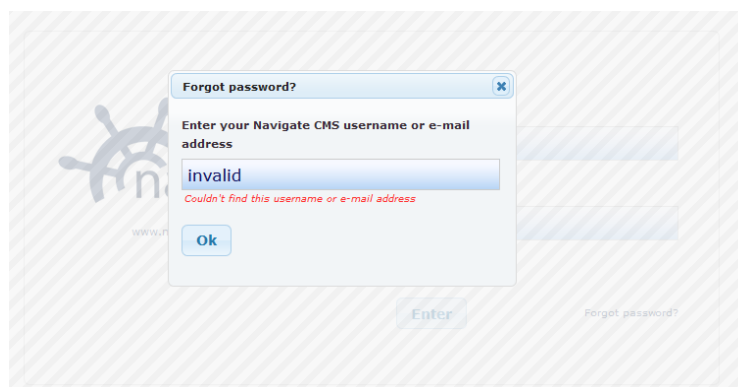
Navigate CMS login page with Forgot password? link.

In order to use this feature the user supplies either their username associated with their account, or the email address which is associate with their account.



Forgot password feature.

When entering the username or email address of a non-existent user, it produces an error stating that the user couldn't be found:



Result of providing details of non-existent user.

When reviewing the HTTP response it can clearly be seen that the HTTP response responds with the message *not_found* in the response body:

HTTP request:

```
POST /navigate/login.php HTTP/1.1
Host: 192.168.0.130
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0) Gecko/20100101
Accept: */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
```

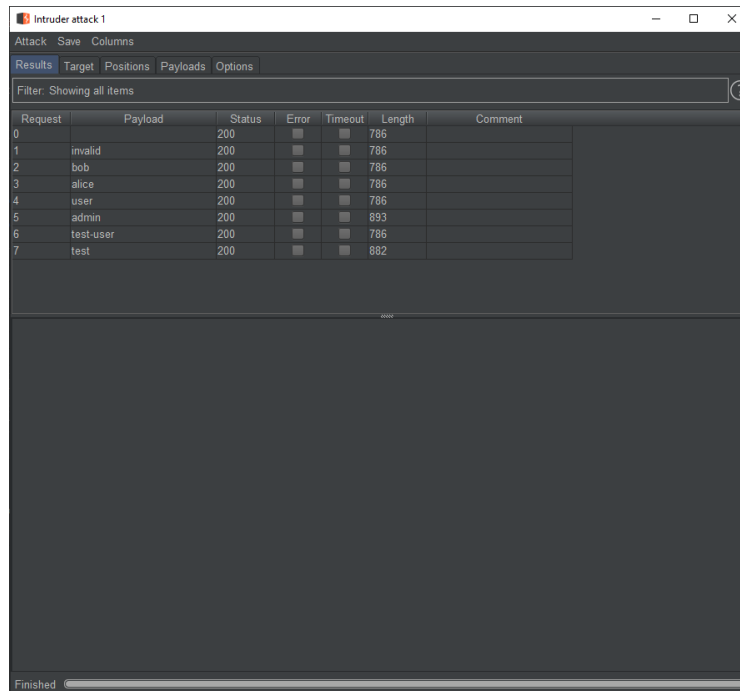
```
X-Requested-With: XMLHttpRequest
Content-Length: 36
Origin: http://192.168.0.130
Connection: close
Referer: http://192.168.0.130/navigate/login.php
Cookie: PHPSESSID=61s0ko8kgh9fikeqt0a78cmp0t; navigate-tinymce-scroll=%78%7D; f
action=forgot-password&value=invalid
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Wed, 10 Jun 2020 16:55:33 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: NVSID_ff0d48a629500e15=61s0ko8kgh9fikeqt0a78cmp0t; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: NVSID_ff0d48a629500e15=61s0ko8kgh9fikeqt0a78cmp0t; expires=Wed, 10
Set-Cookie: PHPSESSID=61s0ko8kgh9fikeqt0a78cmp0t; expires=Wed, 10-Jun-2020 17:
X-Csrf-Token: 236c7e771d9d104865dd84a095910bdeef4434443f7c55be1cdf82bb6037b7f
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 9

not_found
```

Using this one can either use something such as a wordlist to try enumeration which users exist on the system:



The screenshot shows the 'Intruder attack 1' window with a table of results. The table has columns for Request, Payload, Status, Error, Timeout, Length, and Comment. The results show a series of requests with payloads like 'invalid', 'bob', 'alice', 'user', 'admin', 'test-user', and 'test', all resulting in a 200 status code. The interface includes tabs for Results, Target, Positions, Payloads, and Options, and a filter bar at the top.

Request	Payload	Status	Error	Timeout	Length	Comment
0		200			786	
1	invalid	200			786	
2	bob	200			786	
3	alice	200			786	
4	user	200			786	
5	admin	200			893	
6	test-user	200			786	
7	test	200			882	

User enumeration of the Forgot password? feature.

Session Information Leakage - CVE-2020-14017

Navigate CMS stores session information in plaintext files on the server in the

prevent access to this directory and files. However it appears that this doesn't prevent access with later versions of Apache HTTPD.

Depending how the server has been configured, it could be possible to view the contents of the directory:

[illegible]

Directory list of the directory private/sessions which contains all session files

Upon navigating to one of the session files (most preferably the most recent), one is able to view details and sensitive information (such as CSRF tokens) which are associated with the session:



The screenshot shows a web browser's address bar with the URL `192.168.0.130/nginx/private/`. Below the address bar, the full session ID is visible in the page source or developer tools: `carf_session=641d2a3533f5d4c909f47425114642c0f5e43a38d71c45e4504170d170e3f;carf_session_id=1139170299;request_session=32740f9c72abcf2e42e4424f924e3a246f33;nginxgate_proxycookie=...`

Details of session file

If directory listing has been disabled on the webserver (such as via Apache configuration), an attacker is still able to access these files without authentication, however they would need to bruteforce the session IDs to find existing sessions and thus the existing session files:

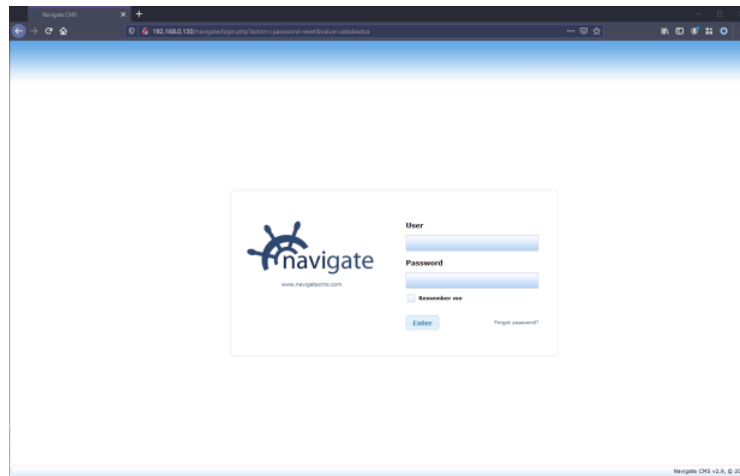
[illegible]

Bruteforcing of session file:

User Password Reset Vulnerability - CVE-2020-14015

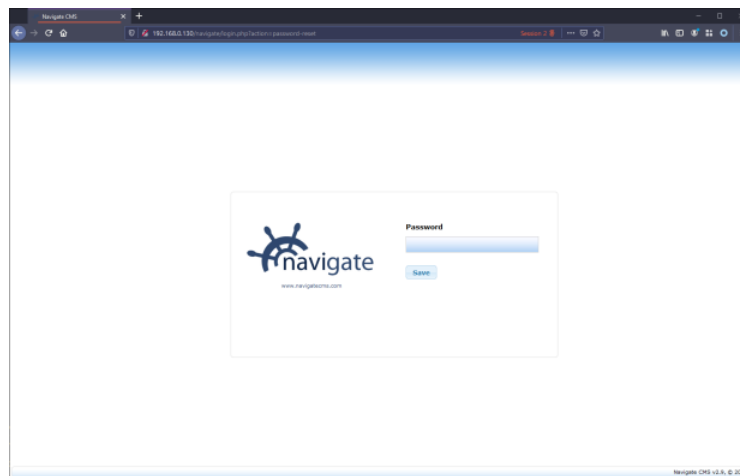
an email is sent the user which contains a password reset link. This link contains an *activation code*, which allows the user to then reset the password associated with their account.

When providing an invalid code, the user is presented with the login screen:



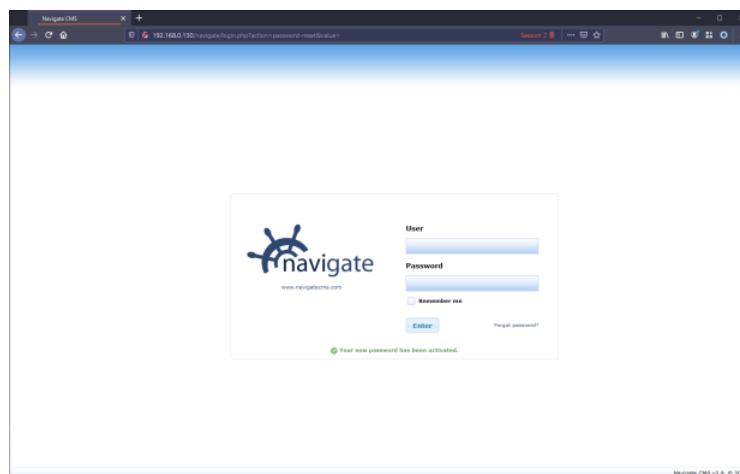
Login screen with invalid password activation code.

However a flaw exists when no code is presented, the user is prompted for a new password:



Password reset for now password activation code.

And setting the password results in success:



Password reset successful.

Upon further investigation, the reason is that when a user is created, their account is created without an activation code:

Database after user created.

When no code is supplied in the password reset process flow, the first user without an activation key will be matched and thus have their password reset.

Stored Cross-Site Scripting - CVE-2020-14018

While there is HTML encoding on *User* field in the Users / Edit page, there is not sufficient encoding on the *E-Mail* field, these leaves it vulnerable to a store XSS vulnerability:

Stored XSS from E-Mail field on the Users / Edit page.

As stated the *User* field is appropriate HTML encoded, preventing any XSS vulnerability from this field on this page:

HTML encoding of the User field.

However the E-Mail field is does not have sufficient HTML encoding, resulting in a stored XSS vulnerability:

XSS vulnerability in the E-Mail field.

This stored XSS vulnerability is persisted and present from both the *User* as well as the *E-Mail* fields when viewing users on the *Users* page:

Stored XSS from the User field.

Stored XSS from the E-Mail field.

Reflected Cross-Site Scripting - CVE-2020-14014

On the landing page once the user authenticates, there is a resource */nagivate.php* which can take the query parameter *fid*. The value from this parameter is reflected back on the page, without having appropriate validation or HTML encoding. This makes the page vulnerable to reflected XSS:

Reflected XSS from the vulnerable fid query parameter.

This is since the value is reflected to a link in the top right corner of the page:

Reflected link contain the reflected XSS.

Resolution

Fixed as of version v2.9.1 r1487 (2020/06).

Vendor Notification

- 10 June 2020 - Initial contact with the vendor, indicating wish to share findings with vendor before disclosing publicly.
- 11 June 2020 - Received response from vendor expressing wish to obtain information about findings.
- 11 June 2020 - Forwarded findings to vendor.
- 17 June 2020 - Vendor shared fixes with myself and asked me to review the fixes.
- 21 June 2020 - Reviewed fixes and confirmed with vendor.
- 22 June 2020 - Vendor confirmed new version and approved public disclosure

Sean Wright

Personal blog of application security advocate, blogging about application security related topics, focused primarily on web based applications.

Navigation

Home
About Me
Events
Findings/Vulnerabilities
Videos
Talks
Media

Security Resources
Resources
Authory
Analytics
Photography

Subscribe

SUBSCRIBE

©2022 Sean Wright. Published with Ghost & Nikko.