Burninator Sec

This blog is about the educational (and sometimes entertainment) value of simple hacks. For active vulnerabilities, real names are concealed.

Wednesday, July 28, 2021

OnyakTech Comments Pro - Broken Encryption and XSS CVE-2021-33484 and CVE-2021-33483

Broken Encryption / User Spoofing (CVE-2021-33484)

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33484

This exploit involves downloading an DotNetNuke module installer for OnyakTech Comments Pro 3.8 and de-compiling it with a tool like JustDecompile. NOTE: it is no longer available for download to my knowledge.

Comments Pro is used for adding comment section functionality to a site.

After decompiling the installer, I find that one of the code files has an intriguing name like "encryption". This has an IV vector hardcoded in it, woo!

But where is the encryption key? We need both in order to do a nefarious enough POC. Well, luckily the requests made to the "CommentsService.ashx" endpoint involve two values, one of which is a JSON field called "key" and one called "displayname". Both appear to be encrypted:

```
{

'key':'jxc+ ... ||',

'atchid':'2080',

'userid':'sH8uVoo..|'

'id':'212',

'commentid':'212',

'displayname':'BhX7vunA8 ... BCNaG8sHo|',

'comment':'definitely fine don't worry about it',

'func':'addcomment'

}
```

I notice that when I throw junk values into the "displayname" value, it will throw an error like "Encryption: The input is not a valid Base-64 string", which is displayed where my display name should be:

This tells me I may be able to control decryption from the client side. So, if I wanted to decrypt it to see what the value of the key is - and I sure do - then I can make that the new value for "displayname" and, voila, there's the key displayed on the page!

Twitter

@burninatorsec

Disclaimer

Information in this blog is for educational purposes only. I am not liable for damages or illegal activity caused directly or indirectly based on the information shared here.

Archive

Now that I have the IV, the key, and even the functions in the source code that show how the encryption and decryption is done, let's use it to do something we're not supposed to do. The goal : to spoof users. Even though the application required a login for most areas, this module seemed to ignore it, so I was able to add (spoofed) comments or add/delete my own or others' comments without authentication. By combining these issues with an unrelated user enumeration issue in DotNetNuke, I can encrypt any user's name and their user ID in the request to spoof a given user. It will even pull in their actual profile image (based on their user ID), so it will look legit.

I recently went to get beer with my local DEFCON group *(in person - vax for hax!)* When I described this out loud, I realized I was having a hard time thinking of a remediation for this type of attack in general. After all, "where to hide the encryption iv/ keys?" is an old problem. But the reverse engineer I was talking to mentioned that the Windows API has it's own encryption that an app could use. I really liked the idea, because it moves control to a deeper layer, to the OS instead of the app. In this particular case, I didn't compromise the server, so the trick of de-compiling would, theoretically, have been foiled by a move like that.

Stored XSS (CVE-2021-33483)

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33483

In the request to the "add comment" endpoint described above, drop in a double {{ to escape the JSON for your XSS payload. When another user visits the page containing the comment with the payload, it will execute.

i.e.

```
{

‘key’:‘jxc+ ... ||’,

‘atchid’:’2080’,

'userid':'sH8uVoo..|'

‘id’:’212’,

‘commentid’:’212’,

'displayname':'BhX7vunA8 ... BCNaG8sHo|',

'comment': '{{ <sCript>prompt(800)</sCript>',

‘func’:’addcomment’

}
```

Posted by burninator at 8:43 PM

Labels: application logic, attack chaining, broken access control, CVE, decompiler, encryption, trial, XSS

No comments:

Post a Comment

To leave a comment, click the button below to sign in with

Newer Post | Home | Older Post

Subscribe to: Post Comments (Atom)