11 NOVEMBER 2020 / RESEARCH

What's in a (re)name: RCE Hunting in CMSs via Unrestricted File Upload

> *What's in a name? That which we call a rose*
> *By any other name would smell as sweet*

- William Shakespeare

Introduction

During a recent bug hunting binge I discovered my first two vulnerabilities that could be exploited to achieve remote code execution (RCE). No bragging rights were earned though, because finding and exploiting these issues was incredibly straightforward. I'm not humble bragging here (I wish). In fact, the issue underlying both vulnerabilities, which each affect a different content management system (CMS), is very basic and was literally the second thing I checked for. This article combines my write-ups for the two RCE vulnerabilities as well as a third issue I found in one of the apps:

- Unrestricted file upload in FlexDotnetCMS v1.5.8 and prior (CVE-2020-27386)
- Unrestricted file upload in HorizontCMS 1.0.0-beta and prior (CVE-2020-27387)
- Incorrect access control in FlexDotnetCMS v1.5.10 and prior (CVE-2020-27385)

P.S. While working on this here blog thingy, my old fiction instincts kicked in and I ended up penning a little story meant to serve as a metaphor for the two RCE flaws. This creation makes up the final section, so in case infosec-oriented allegories aren't your thing, feel free to skip it, ~~you uncultured swine~~.

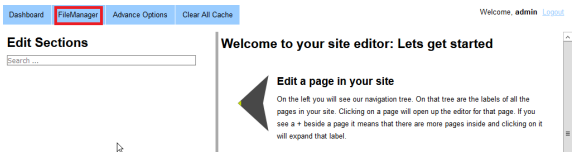
P.P.S. All of these issues also affected Internet-facing demo apps. =O

Unrestricted File Upload: You were always on my mind...

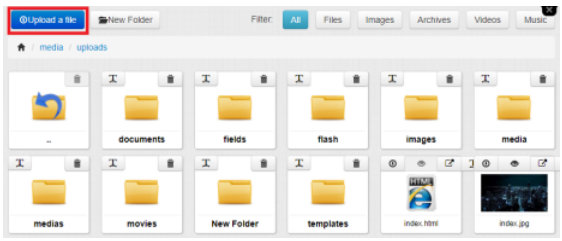
FlexDotnetCMS and HorizontCMS are both open source projects with several dozen stars on GitHub, which is how I found out about them. When I started testing these apps, I had just contributed a Metasploit exploit module that took advantage of a very basic arbitrary file upload vulnerability in a PHP web app. Because this attack was still fresh in my mind, it was the first thing I checked for in both applications, as you will see below.

Unrestricted file upload in FlexDotnetCMS (CVE-2020-27386)

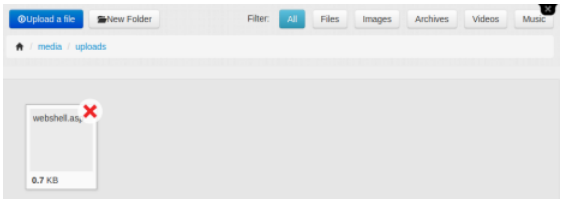
The first app I tested was FlexDotnetCMS. The functionality of the app is reserved for authenticated users, so I started by logging in as the admin user. Once the user dashboard loaded, the *FileManager* immediately caught my eye, so I opened it.



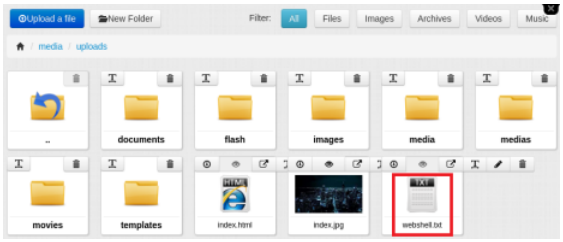
As you would expect, the *FileManager* allows users to upload, edit, view and delete files. So it seemed like a great place to start my investigation.



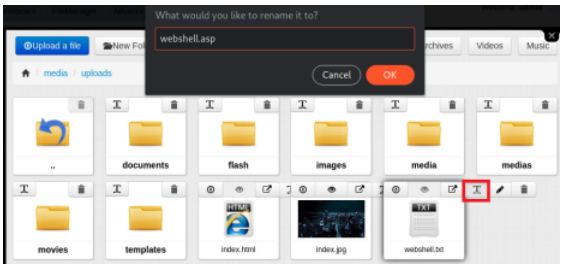
FlexDotnetCMS is an ASP .NET application, so I wanted to see if it would be possible upload a rogue ASP file. On GitHub I found a simple ASP web shell that would allow me to execute code on the server if I could find a way to upload and access the file. I grabbed the file from GitHub, removed a bunch of unnecessary code from it, and saved it as *webshell.asp*. I tried uploading this file, but FlexDotnetCMS didn't let me because the ASP file extension was blacklisted.



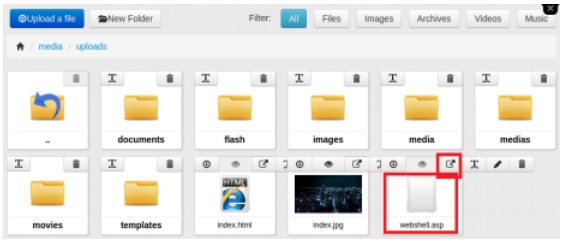
Alright, so the app had at least the most basic protection against arbitrary upload of files with dangerous extensions. But what about files with seemingly innocent extensions? In order to test this, I renamed my payload to *webshell.txt* and tried to upload it.



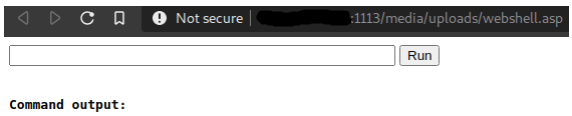
It worked! My file appeared in the uploads folder and I was able to access the contents via a direct URL to the file. However, the server currently treated my payload as a text file, while I needed it to regard it as an ASP file so it would execute my web shell instead of merely displaying the code. The most straightforward way of doing this would be to rename the file. I noticed that FlexDotnetCMS had a simple rename function that would let me test this.



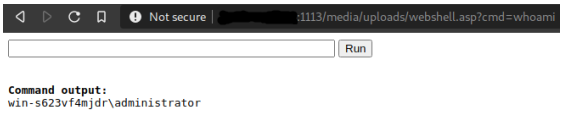
I clicked the rename function, changed the file name back to *webshell.asp*, hit enter, and...



Success! Apparently, FlexDotnetCMS only blacklisted dangerous file types during upload, but not when users renamed files, which is like locking your car but leaving the window wide open. In order to verify that I was actually able to access the web shell now that it was on the server, I clicked on the button redirecting to the file URL at /media/uploads/webshell.asp. The result?



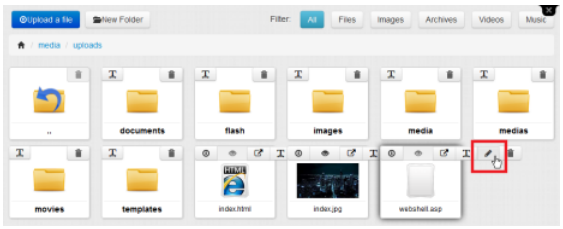
Behold, our web shell! Now let's see if we can get actually use it to run some code. I tried running the *whoami* command, which should display the username of the active user on the server...



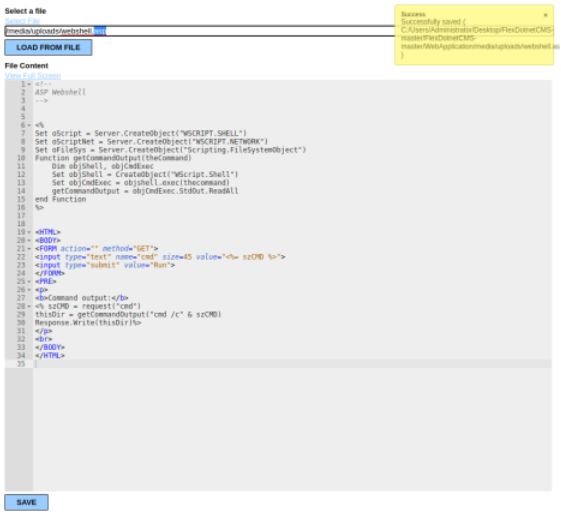
BOOYAH! Remote code execution achieved!

Well, that was easier than I thought. And the best part was that the attack resulted in remote code execution with *elevated privileges*. FlexDotnetCMS was designed to run on a Microsoft Internet Information Services (IIS) server. If IIS or IIS Express is configured to allow remote connections, the IIS/IIS Express process must run with elevated privileges, and an attacker exploiting this vulnerability will inherit these privileges from the server process.

But there was more. I went back to the *FileManager*, and opened my webshell in the *FileEditor* (*/Admin/Views/FileEditor/Default.aspx?LoadFile=/media/uploads/webshell.asp*).



I discovered that in addition to letting users edit files, the *FileEditor* offered a second method for renaming files. Testing with the *webshell.txt* file confirmed that the attack was also possible by using the *FileEditor* instead of the rename function to rename the payload.



Demo video

The demo video below shows the full attack via the rename function:

Metasploit module

Instead of throwing together a proof-of-concept (PoC) exploit with limited functionality, I decided to write a full-fledged Metasploit module for this attack.

Patches

- FlexDotnetCMS v.1.5.8 fixes the attack via the rename function, but is still vulnerable to renaming via the *FileEditor*.
- FlexDotnetCMS v.1.5.9 fixes the attack via the *FileEditor* as well.

Incorrect access control in FlexDotnetCMS (CVE-2020-27385)

While scrutinizing the *FileEditor*, I discovered a second vulnerability, which was an access control issue. As it turns out, the *FileEditor* allowed users to read and modify files from anywhere on the host system. There were two ways to achieve this. The first method was by entering the full path to an existing file into the *FileEditor*'s "Select File" input field and then loading the file.

The second method was via directory traversal, i.e. by entering a .. (dot dot) path such as *../../<file>* or *..\..\..\<file>* in the input field.

While both of these attacks require the attacker to guess the correct full path or traversal path to an existing file, they could both be used to enumerate the host system in search of sensitive files to access and even modify. And because these actions were performed with elevated privileges (inherited from the IIS/IIS Express process as mentioned above), an attacker would even have access to files in protected directories like *Program Files*, *Program Files (x86)*, *Windows\System32* and *Windows\SysWOW64*.

Demo video

The demo video below shows how this attack works via directory traversal:

Patches

- FlexDotnetCMS v.1.5.8 fixes the attack when users specify the full path in the *FileEditor*, but is still vulnerable to directory traversal.
- FlexDotnetCMS v.1.5.11 fixes the attack via directory traversal as well.

Unrestricted file upload in HorizontCMS (CVE-2020-27387)

After wrapping up my research on FlexDotnetCMS, I continued my hunt for RCE vulnerabilities, and soon came across HorizontCMS. Much like with the former app, all core functionality is only available to authenticated users. After logging in to the user dashboard, I soon found that HorizontCMS also has its own *File Manager*, so I started my research there.

Since HorizontCMS is a PHP app, I wanted to see if the *File Manager* would allow me to upload and execute a PHP payload. In order to test this, I created a PHP file with the following contents:

```
<?php system($_GET["cmd"]); ?>
```

When I attempted to upload this mini PHP web shell as *test.php*, I was surprised to see that it seemed to work straight off the bat.

However, when I used the *File Manager* to look for my payload, I noticed that the file had been renamed to a .TXT file with a long, randomly generated name.

I obtained the full URL to the file by hovering over the download button. While I was able to access the file at */storage/<file_name>*, I could only view the contents, since HorizontCMS was simply treating it as a text file. Much like the person in the below images, I attempted in vain to overcome this insurmountable obstacle, and eventually threw in the towel.

Just kidding. I simply located the *File Manager*'s rename function and tried changing the name back to the original.

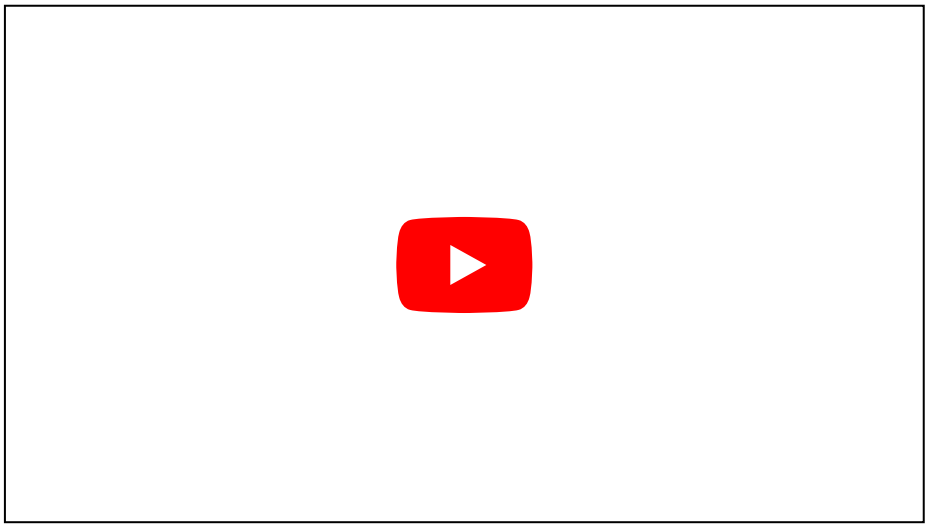
As you've probably guessed by now...

It worked like a charm! A visit to */storage/test.php* now displayed a blank page, as expected. Finally, I attempted to run the Unix *id* command (since I had installed HorizontCMS on an Ubuntu VM). The result...

Yay! RCE again!

Demo video

The demo video below shows the full attack:



Metasploit module

Once again, my PoC for the attack is a Metasploit module.

Patch

Commit *436b5ab* fixes the vulnerability.

Implications: A sea of vulnerabilities

Precisely because the vulnerabilities discussed here are far from impressive attacks, they point to a much larger problem that continues to plague software development: negligence of basic security practices. Unfortunately many developers still don't pay sufficient attention to security. As a result, GitHub, SourceForge and other open source platforms are sprawling with apps vulnerable to severe, low-complexity zero-day attacks like those covered here. I'm not going to pretend this is a solvable problem at this point, though I do think significant progress can be made in the coming years as the result of growing security awareness and the availability of an increasing number of automated code analysis solutions, such as GitHub's new Code Scanning feature.

On the off chance that anyone reading this is interested in a few pointers on secure coding, here are my two cents:

- **Adopt a secure coding mindset**: Assume that people are going to try and find vulnerabilities in your software, no matter how insignificant the project may seem.
- **Follow best practices**: Follow secure coding best practices such as those outlined in the OWASP Secure Coding Practices-Quick Reference Guide.
- **Familiarize yourself with common attacks**: Getting familiar with the OWASP Top 10 Web Application Security Risks and other common attacks will help you avoid writing vulnerable code.
- **Scan your code:** If possible, use automated code scanning solutions to help catch at least the most basic vulnerabilities in your code.

Alright folks, it's finally time for the highlight of this article: my little metaphor-story-thingy. Enjoy!

A Tale of Two Chainsaws

(or one, but like, in different stages of assembly)

Dawn

"Hold it! I cannot let you enter with that," the security guard outside the hardware store says sternly as he moves his hand toward the banana-yellow taser in his duty holster.

“Why?” W. wonders with feigned surprise, seemingly unimpressed by the guard’s aggressive body language.

“Because you are holding a running chainsaw, and this ain’t Texas,” the guard retorts in a low, threatening voice, while placing his right hand firmly over the taser’s handle.

“Fair enough,” W. replies, casually shrugging their shoulders before flicking the switch to kill the engine. “May I enter now?”

“You most certainly may not!” The guard bellows. “We don’t sell chainsaws, so I don’t know what the hell you’re planning to do with it, but it ain’t gonna happen on my watch. Beat it, freak, before I tase the life out of you.”

“If you say so,” W. replies, curling their lips into a sly grin. “Have a nice day.”

Dusk

“Good evening, may I inspect the contents of your bag?” The guard demands rather than asks as W. walks up to the store.

“Of course,” W. nods, handing over the duffel bag.

“Is this… Is this a chainsaw?” The guard inquires with a suspicious frown.

“It was, before I dissembled it,” W. responds calmly. “It broke down, so I took it apart in an attempt to fix it, but that didn’t work, and now I don’t know how to put it back together again. I brought it here because I’m hoping to get some advice on how to fix it. Plus I’m pretty sure I lost a few nuts and washers in the process that I’ll need to replace.”

“Sounds like me whenever I’m asked to fix something,” the guard chuckles. “Have a good one.”

“Thanks! Same to you.”

Once inside, W. grabs a shopping cart and puts the duffel bag in it. Leisurely strolling through the aisles, they fill their cart with a variety of items. A screwdriver, a few combination wrenches, a chainsaw sharpener, a bottle of bar & chain oil, a pair of safety gloves and finally a pair of safety glasses. To a casual observer, W. would appear like a regular customer. Nothing out of the ordinary. However, that casual observer would soon be proven wrong.

W. heads over to the garden section and parks the cart next to a large picnic table. No one seems to take notice of W. as they take the items out of the shopping cart and place them neatly on the wooden tabletop. W. sits down at the table, zips open the duffel bag, and begins to assemble the chainsaw with the speed and precision of a skilled mechanic. A few passing shoppers and employees raise their eyebrows at the rather impressive spectacle, but no one says a word until W. picks up the chainsaw and pulls the starter rope, bringing the engine to life with a loud sputter.

“Hey! What are you doing? Are you crazy?!” A voice yells from behind. W. turns around to see the security guard running toward them from across the store, two employees in bright orange uniforms at his heels. “You can’t use that thing in here!” The guard shrieks with rage.

“Well, it seems like I fixed it,” W. responds coldly. They flick the safety switch and press the trigger for half a second, letting the chain race around the blade with a deafening roar. “See? It runs” W. adds, as the noise dies down. Holding the chainsaw at the handle with one hand, W. uses the other to put the safety glasses on, never breaking eye contact with the guard, who’s bloated face is slowly turning pale. “It runs,” W. repeats in a dark, ominous voice, moving their hand back toward the trigger. “And perhaps so should you…”

Stay Informed

- Connect with us on Linkedin for Professional Security Tips
- Like us on Facebook for Personal Security Tips
- Follow us on Twitter for News & Threat Updates

Erik Wynter
Erik Wynter is a junior pentester, Metasploit contributor and script kitty.

Read More

— Vonahi Security's Blog —
research

When the PATH to SYSTEM is wide open: Philips SmartControl DLL hijacking (CVE-2020-7360)

2019/2020: A Few Cybersecurity Reflections and Predictions

13 Freaky Infosec Facts

See all 8 posts →

PRIVILEGE ESCALATION

SrClient DLL Hijacking: a Windows Server 2012 0-day that won't be patched

This blog discusses a DLL hijacking vulnerability affecting all versions of Windows Server 2012 (but not Server 2012 R2). This 0-day vulnerability can be exploited for privilege escalation by any regular user and does not require a system reboot, yet it will not be patched by Microsoft.

11 MIN READ

RESEARCH

When the PATH to SYSTEM is wide open: Philips SmartControl DLL hijacking (CVE-2020-7360)

Earlier this year our threat researcher found a DLL hijacking flaw affecting Philips SmartControl (CVE-2020-7360). Our latest blog post combines a write-up of this vulnerability with a general introduction to DLL hijacking for infosec students.

14 MIN READ