EASYII CMS

REMOTE CODE EXECUTION

EASY RCE: Unrestricted File Upload and Remote Code Execution Exposed

Critical CVE-2022-3771

Discovered by HADESS

27 July 2023

HADESS

WWW.HADESS.IO

Executive Summary

1. Unrestricted File Upload Vulnerability (CVE-2022-3771): The first vulnerability allows attackers to perform unrestricted file uploads through the function `Upload::file` in the `helpers/Upload.php` file of the File Upload Management component. Attackers can exploit this vulnerability remotely, potentially leading to unauthorized access, data manipulation, and disruption of the system's availability. An exploit for this vulnerability is known, making it crucial for users to update their easyii CMS installations to a patched version.

2. Remote Code Execution (RCE) via File Upload: The second vulnerability was discovered in the same `getFileName` function within `helpers/Upload.php`. This vulnerability allows an attacker to manipulate the uploaded filename and append a malicious file extension, such as ".php." If the application fails to properly validate the file type and blindly concatenates the extension, the uploaded file could be executed as PHP code by the web server. This results in potential RCE, giving the attacker full control over the system.

01

Advisory

File upload functionalities are commonplace in modern web applications, allowing users to upload various types of files, such as images, videos, and documents. However, the seemingly innocent act of uploading files can potentially open a pathway for devastating security risks, one of the most critical being Remote Code Execution (RCE).

Remote Code Execution occurs when an attacker exploits vulnerabilities in a web application's file upload mechanism to execute arbitrary code on the targeted server. This can lead to a complete compromise of the system, granting unauthorized access and control over sensitive data, server resources, and even the entire hosting environment.

One of the common ways RCE is facilitated through file upload is by manipulating the uploaded file's filename and extension. When an attacker uploads a file, they might craft the filename to end with a malicious extension, such as ".php," thereby disguising the file as a legitimate one, like an image or document. If the application fails to properly validate the file type and blindly concatenates the uploaded file's extension to the base filename, the web server could interpret the uploaded file as a PHP script.

As a result, when a user or the web application accesses the supposedly benign file, it gets executed as PHP code by the server, potentially leading to disastrous consequences. The attacker gains unauthorized access to the server environment and can perform various malicious activities, ranging from data theft and defacement to taking full control of the system.

In this article, we will explore the intricacies of Remote Code Execution when facilitated through file upload vulnerabilities. We will analyze the common mistakes made in file upload implementations, discuss real-world examples of RCE incidents, and provide best practices and security measures to safeguard web applications against such attacks. Understanding the risks associated with RCE via file upload is crucial for developers and system administrators, enabling them to fortify their applications and infrastructure against potential exploitation. By adopting proactive security measures, organizations can ensure the integrity and security of their web applications, protecting themselves and their users from the threats posed by Remote Code Execution.

Abstract

EASYII CMS

Content Management Systems (CMS) are widely used to manage, create, and organize digital content for websites and applications. One of the essential functionalities of a CMS is the ability to upload files, such as images, videos, and documents, to enrich the website's content. However, this seemingly benign feature can become a significant security risk if not implemented carefully.

Remote Code Execution (RCE) is one of the most critical security vulnerabilities that can emerge when handling file uploads in a CMS. RCE occurs when an attacker can execute arbitrary code on a target system remotely, effectively gaining unauthorized access and control over the web server and its underlying infrastructure. When RCE is combined with file upload functionality, it can result in severe consequences, including data breaches, defacement of websites, and even full compromise of the entire hosting environment.

In this context, it is essential to understand the potential pitfalls of file upload implementations in CMS platforms. This article delves into the intricacies of RCE vulnerabilities arising from file upload functionalities in CMS and explores the common security pitfalls that developers must be aware of. Additionally, we will discuss best practices and mitigation strategies to secure CMS file upload functionality against RCE attacks, ensuring the integrity, confidentiality, and availability of the web application and its hosting infrastructure. By gaining a comprehensive understanding of RCE vulnerabilities in CMS file uploads, developers and administrators can take proactive measures to safeguard their systems and protect against malicious exploitation.

02

Technical Analysis

Technical Analysis

File Upload Functionality: The `file` function within the `Upload` class is responsible for handling file uploads. It takes an `UploadedFile` instance, which likely represents the file being uploaded, and optional parameters for the directory and name postfix.

2. File Upload Directory: The variable `UPLOADS_DIR` holds the directory where the uploaded files will be stored. The directory is relative to the web root, as indicated by the usage of `Yii::getAlias('@webroot')`.

3. File Upload Process: The function `file` first generates a unique filename using the `getFileName` function. It then combines the filename with the upload directory to form the complete file path. After that, it attempts to save the uploaded file using the `saveAs` method of the `UploadedFile` class.

4. Potential RCE Vulnerability: The potential RCE vulnerability lies in the `getFileName` function. This function is responsible for generating the filename for the uploaded file. It takes the `UploadedFile` instance and an optional `namePostfix` parameter.

The issue is with the `$fileInstanse->name` variable, which is directly used to construct the `$baseName`. If the `name` property of the `UploadedFile` instance is controllable by an attacker, they can inject malicious input containing path traversal sequences or even PHP code.

Let's break down the vulnerability:

- The `UploadedFile` instance is likely populated from user input during the file upload process, which can be controlled by an attacker.
- If the attacker is able to control the `name` property, they can craft a filename that contains "../" or other path traversal sequences, allowing them to escape from the intended upload directory.
- By including PHP code in the filename, the attacker can potentially achieve remote code execution. This is because the uploaded file is later saved with the filename generated by the `getFileName` function.

For example, if the attacker uploads a file with the name `malicious_file.php`, the resulting filename could be something like `uploads/malicious_file-<random_string>.php`. As a result, the uploaded PHP file will be accessible on the server and could be executed if the server is not properly configured.

The vulnerability arises from the insecure construction of the filename in the `getFileName` function. It fails to properly validate and sanitize the input provided as the `$fileInstanse->name`, which is the name of the uploaded file.

```
static function getFileName($fileInstanse, $namePostfix = true)
{
       $baseName  =  str_ireplace('.'.$fileInstanse->extension,  '',  $fileInstanse->name);
    $fileName =  StringHelper::truncate(Inflector::slug($baseName), 32, '');
    if($namePostfix || !$fileName) {
         $fileName .= ($fileName ? '-' : '') . substr(uniqid(md5(rand()), true), 0, 10);
    }
    $fileName .= '.' . $fileInstanse->extension;

    return $fileName;
}
```

Vulnerability Explanation: The `getFileName` function tries to create a base filename by removing the file extension from the uploaded file name (`$fileInstanse->name`). However, it does not adequately validate and sanitize the input.

If an attacker can control the uploaded filename, they can exploit this vulnerability in two ways:

a) Path Traversal: An attacker can craft the filename to include path traversal sequences like `"../"` to traverse outside the intended upload directory (`UPLOADS_DIR`). This could potentially allow the attacker to overwrite sensitive files on the server or access files outside the designated upload directory.

b) Remote Code Execution (RCE): The function concatenates the extension of the uploaded file (`$fileInstanse->extension`) to the base filename. If the attacker uploads a file with a malicious PHP script and manipulates the filename to end with `.php`, the uploaded file may be executed as PHP code when accessed through the web server.

Mitigation:

To mitigate this potential RCE vulnerability, it is essential to properly sanitize and validate user-controlled input, especially when constructing filenames or paths. Specifically:

1. Avoid using user-controlled data directly to construct filenames. Use a whitelist approach to only allow specific characters or patterns in filenames.
2. Sanitize and validate the `name` property of the `UploadedFile` instance to prevent path traversal and other malicious input.
3. Consider using a secure file upload library that automatically handles file naming and storage to reduce the risk of vulnerabilities like this.

To reproduce the Remote Code Execution (RCE) vulnerability in the getFileName function, an attacker needs to carefully craft the uploaded file name and utilize a PHP payload in the filename. Here's a step-by-step guide on how an attacker can construct the payload to achieve RCE:

Step 1: Prepare the PHP Payload The attacker needs to prepare a PHP payload that will be injected into the filename. The payload is a snippet of PHP code that, when executed, will give the attacker control over the server. An example payload is as follows:

<?php echo system($_GET['cmd']); ?>

This PHP payload simply executes a system command passed via the 'cmd' parameter in the URL and echoes the output.

Step 2: Craft the Filename Now, the attacker must craft the filename to include the PHP payload at the end. For instance, the attacker may choose the following filename:

malicious_file.php<?php echo system($_GET['cmd']); ?>

Step 3: Upload the File The attacker proceeds to upload the file with the crafted filename. The CMS invokes the getFileName function during the file upload process.

Step 4: Filename Processing The getFileName function processes the uploaded filename

- The function removes the file extension from the uploaded filename.
- It then generates a base filename using Inflector::slug to create a URL-safe version of the file name.
- If the $namePostfix parameter is true or if the base filename is empty, a postfix is added to the filename to ensure uniqueness.
- Finally, the function appends the original file extension back to the filename.

Step 5: Exploiting the Payload Since the uploaded filename contains the PHP payload, the generated $fileName will be:

When this file is accessed via the web server, the PHP payload will be executed, allowing the attacker to execute arbitrary system commands. For instance, the attacker can execute commands by accessing the file in the browser with the 'cmd' parameter:

http://example.com/uploads/malicious_file.php?cmd=ls

This will execute the ls command on the server and return the output to the attacker.

03

Conclusion

In this discussion, we delved into the technical analysis of critical vulnerabilities found in easyii CMS, focusing on two significant security issues: Unrestricted File Upload and Remote Code Execution (RCE) via File Upload. These vulnerabilities posed substantial risks to the confidentiality, integrity, and availability of web applications utilizing the CMS.

The Unrestricted File Upload vulnerability allowed attackers to upload malicious files without any restrictions, potentially leading to the execution of arbitrary code, unauthorized access, and data manipulation. The existence of an exploit for this vulnerability emphasized the urgency for users to update their easyii CMS installations to the latest patched version, safeguarding their systems against potential exploitation.

The RCE via File Upload vulnerability emerged from improper handling of filenames during the upload process. Attackers could manipulate the filename to include a malicious PHP extension, tricking the web server into executing the uploaded file as PHP code. This flaw granted unauthorized access and control over the server, posing serious security threats to the application and its hosting environment.

In light of these vulnerabilities, it is crucial for CMS administrators and developers to prioritize security measures. Implementing strict input validation, using secure file upload libraries, and regularly updating the CMS to receive security patches are essential steps in mitigating such risks.

Furthermore, organizations must stay vigilant against potential threats, continuously monitoring their web applications and conducting security audits to identify and address vulnerabilities promptly.

By being proactive in strengthening the security posture of CMS installations, developers and administrators can safeguard sensitive data, protect user privacy, and ensure the robustness of their web applications against malicious attacks. As the landscape of cybersecurity evolves, ongoing vigilance and a security-first mindset are essential to stay ahead of potential threats and protect digital assets effectively.

HADESS

cat ~/.hadess

We are "Hadess"; A group of cyber security experts and white hat hackers who, in addition to discovering and reporting vulnerabilities to big companies such as Google, Apple and Twitter, have the honor of working with famous Iranian companies over the past years. HADESS Company provides its customers with integrated solutions in the field of cyber security, with a deep insight and understanding of the software development process as well as the development infrastructure.

Website:

WWW.HADESS.IO

Email

MARKETING@HADESS.IO