Ghost CMS 4.3.2 - Cross-Origin Admin Takeover

BY PAUL GERSTE | AUGUST 31, 2021

Security

Ghost is one of the most popular Node.js-based Content Management Systems (CMS). According to the vendor, there are currently more than 2.5 million installs of it and the project has more than 38k stars on GitHub. During our research on open-source applications, we analyzed the code and found a vulnerability in Ghost 4.3.2 that allows attackers to gain control of admin accounts.

In this blog post, we will first look at some web technologies that are required to understand the vulnerability. Then we will show the vulnerability and how it could have been exploited by attackers. Finally, we will explain how to avoid or fix such issues during development.

Impact

The code vulnerability, CVE-2021-29484, was introduced in Ghost 4.0.0 and fixed in version 4.3.3. It is a DOM-based Cross-Site Scripting (XSS) issue that allows attackers to take over accounts, including admins. This would allow them to read or modify any data on the site.

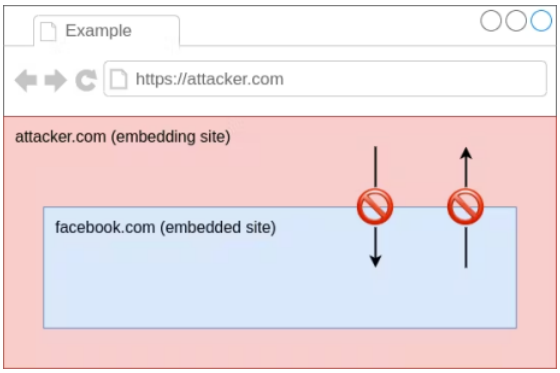
Exploiting this vulnerability requires the victim to visit a malicious link while being logged in to the Ghost admin area. The affected versions of Ghost are vulnerable in the default configuration, and there is no setting to disable the affected component.

The following video demonstrates the exploitation of the vulnerability by having an admin click on a malicious link that creates a new privileged account for the attacker without the victim noticing it:

vulnerability works and how to avoid such issues during development.

Background

When a website embeds another website using an `<iframe>` element, some rules control how both sites can interact with each other. This set of rules is called the Same-Origin Policy (SOP). It prevents different websites from directly reading or modifying each other unless they come from the same *origin*. A website's origin consists of the protocol, the host, and the port of the website's URL. For example, `https://example.com` is the origin of `https://example.com/test/?id=42` (the port is omitted here because it can be derived from the protocol).



As a result, a website on `https://`**attacker**`.com` cannot read or manipulate any data of `https://`**facebook**`.com`, but `https://attacker.com/`**xxx** can access the content of `https://attacker.com/`**yyy**. This is an important security mechanism because otherwise, every website you visit could steal your private Facebook messages or read your bank data, just by embedding the victim site.

There is, however, still a way for websites to communicate with other websites that are *cross-origin*, meaning that they have different origins. To do this, websites can send and receive message events by using the postMessage API. This method is pretty secure by default because it does not allow the sites to directly access each other's DOM, but there is still room for things to go wrong when handling these messages, as we will see in the next section.

DOM-based XSS in Theme Preview (CVE-2021-29484)

Ghost is a CMS that consists of two components. The first component is the page containing the content, the other one is the admin area. The admin frontend is usually served on the `/ghost/` sub-path of a Ghost site, but it can also be served on another domain if users want to go with an extra-secure setup. The admin area allows users with various roles to log in and perform tasks, e.g. writing new blog posts, editing settings, or changing the site's theme.

In Ghost 4.0.0, a theme preview feature was added to the admin frontend. It consists of a static HTML page that is served in the context of the admin area at `/ghost/preview`. The page, simplified for brevity, looks like this:

core/server/web/admin/views/preview.html:

```
1   <script type="text/javascript" charset="utf-8">
2       (function(){
3           function onReceive(message) {
…               // ...
6               document.write(message.data);
…               // ...
9           }
…           // ...
34          if (window.addEventListener){
35              addEventListener("message", onReceive, true);
…               // ...
38          }
…           // ...
42      })();
…       // ...
53  </script>
```

It contains a script that listens for `message` events (line 35). If such an event occurs, its content is added to the page in line 6, but without verifying the event's origin. This constitutes a vulnerability because a site from any origin could send such a message. The whole theme preview component is embeddable from anywhere, as there is no `X-Frame-Options` header and no `frame-ancestors` directive in the `Content-Security-Policy` header that would prevent it.

This allows for DOM-based XSS within the context of the admin panel. An attacker could craft a website that embeds the theme preview page and sends a malicious HTML payload to it. The payload will then be inserted into the page, executing any JavaScript in it. Since the victim is logged in, the attacker can now do anything the victim is authorized to do.

Attackers could use this to take over privileged accounts, such as admins or owners, by luring them into visiting an attacker-controlled website while being logged in. The malicious payload could create a new admin account for the attacker, which would provide unrestricted access to the Ghost admin area. This is demonstrated in the demo video above.

```
  if (event.data === 'loaded') {
    const payload = 'alert(origin)';
    iframe.postMessage(`<script>${payload}<\x2fscript>`, '*');
  }
});
</script>
```

This example exploit works by embedding a Ghost instance's theme preview page in an iframe and then using the postMessage API to send a message that contains a malicious script once the iframe has loaded. When a victim, e.g. an admin user, visits the attacker's page, the script payload is executed and can perform any action as the admin. This would result in the take-over of the Ghost site, as the attacker could read and modify everything on it.

This vulnerability shows that even in high-quality codebases, things can slip through. An automated approach helps to catch issues before they go into production or even before they leave the IDE.

Patch

In this case, the vendor chose to remove the affected component because it was unused anyway. In other scenarios, there might not be such an easy option.

The main issue was that *any* website could have sent a message event and the theme preview component would not validate where it came from. Fortunately, message events have the origin property that can be used to validate the sender. A straightforward fix would be to compare the event's origin with a set of allowed origins and reject any message that comes from somewhere else. Example:

```
const allowedOrigins = [
    'https://example.com',
    'https://blog.example.com',
];
window.addEventListener('message', (event) => {
    if (!allowedOrigins.includes(event.origin)) {
        return;
    }
    handleEvent(event);
});
```

So if your code is handling cross-origin message events, you can check if it uses the messages in potentially dangerous ways, such as inserting unfiltered data into the page. In this case, we recommend checking the origin to verify that events come from non-malicious origins, as shown in the example above.

Timeline

Date	Action
2021-04-27	We send a detailed advisory via email
2021-04-27	Vendor confirms the issue, asks for additional proof-of-concept (PoC)
2021-04-27	We send an additional PoC that demonstrates the impact
2021-04-28	Vendor asks for further clarification
2021-04-28	We provide more details
2021-04-29	Vendor releases version 4.3.3

Summary

In this blog post, we analyzed a code vulnerability found in Ghost 4.3.2, a widely-used open-source CMS written in JavaScript. We outlined how the Same-Origin Policy works, and how unsafe handling of Cross-Origin messages can lead to the takeover of a Ghost instance. We also explained how to prevent vulnerabilities of this kind.

We reported these vulnerabilities to the vendor in late April 2021. They confirmed and fixed the vulnerabilities immediately and took their product's security very seriously, so huge kudos to the Ghost security team! If you are running Ghost, we recommend updating to at least version 4.3.3.

Related Blog Posts

- *Zimbra 8.8.15 - Webmail Compromise via Email*
- *WordPress 5.7 XXE Vulnerability*
- *Grav CMS 1.7.10 - Code Execution Vulnerabilities*

PAUL GERSTE
Vulnerability Researcher