Hacking plugin ecosystems: A complete guide

BY BLACKBIRD-EU · SEPTEMBER 2, 2025

Add-on (or plugin) ecosystems unlock an entire new world of integration possibilities while also complementing the platform's extensibility to developers. However, in practice, finding the right balance between adding extensibility and maintaining security often proves to be difficult. The root cause stems from a lack of following security best practices. Proper isolation is, for instance, never fully adopted, with a common consequence of opening a new attack surface where vulnerabilities could emerge.

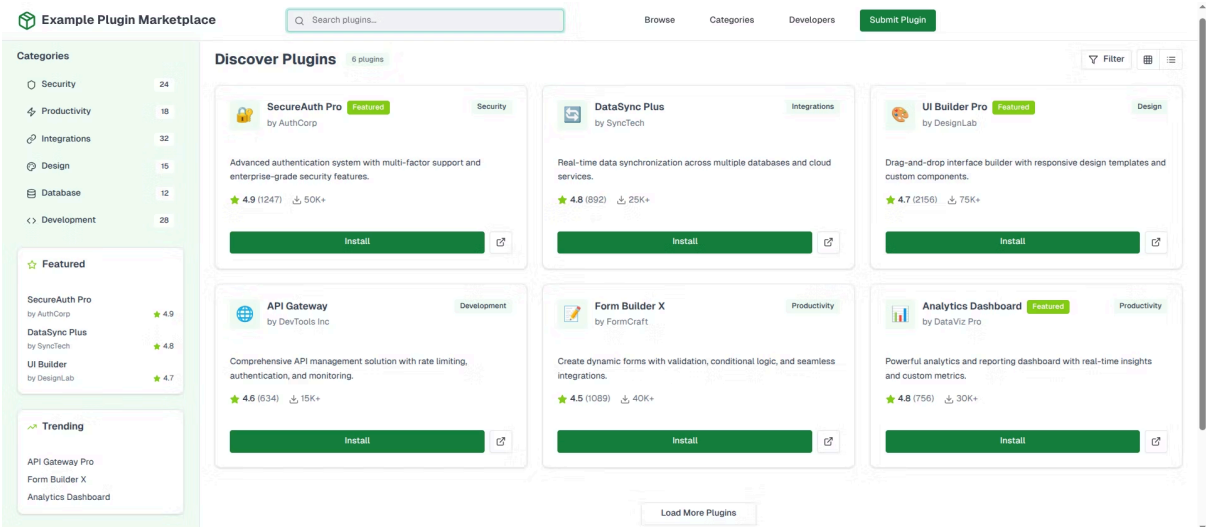
In this article, we will be exploring the most common vulnerabilities to test for in web-based add-on/plugin ecosystems.

Let's dive in!

Identifying targets with add-on/plugin ecosystems

Launching an add-on/plugin marketplace is often the next logical step when it comes to enhancing accessibility, adding extensibility, and introducing endless possibilities of new use cases. We'd want to identify the companies that already have an add-on/plugin marketplace provided to the end user.

This process is simple. All we have to do is browse our target and look for a place where we can enable or install plugins. Sometimes, this is promoted on the landing page as one of the main features of the platform.



Example of an add-on/plugin marketplace

Below are a few platforms and software products that have integrated an add-on or plugin marketplace:

- **GitHub Marketplace** provides plugins and tools that integrate directly with GitHub's platform for developers to automate workflows, CI/CD, code reviews, and more.
- **Atlassian Marketplace** enables users to expand products like Jira, Confluence, and Bitbucket with add-ons and plugins for project management, collaboration, and development workflows.
- **Salesforce AppExchange** is a marketplace for business apps, integrations, and add-ons for Salesforce, a cloud-based customer relationship management (CRM) platform.
- **Obsidian** is a powerful knowledge base and note-taking app that also provides a marketplace for its users to install and use third-party plugins.
- **Discord** is a free messaging service that also supports an extensive bot ecosystem to extend standard features in the messaging app.

Common vulnerabilities in add-on/plugin ecosystems

As the examples from before point out, a plugin ecosystem can open up a whole new world of use cases to the end user. For us, security researchers, it's another attack surface we can explore.

The most common issue developers experience when introducing a plugin marketplace is defining a clear boundary between providing developers with powerful extensibility features to develop new plugins and maintaining overall security to protect the end-user.

Let's look at the 7 most common security risks to test for when your target has a plugin marketplace. We recommend you use this article as a guide, not a strict checklist. Every platform is different, so you'll need to adapt these tests based on what your specific target can do.

1. Inadequate sandboxing & isolation

Sandboxing is a security measure to help isolate plugins from accessing the parent (or host) environment. It helps prevent web-based plugins from reaching protected API endpoints to conduct unwanted actions (such as performing unauthorized network requests), accessing other plugins' data, or even making changes to the host server (in the event server-side code execution is supported), etc.

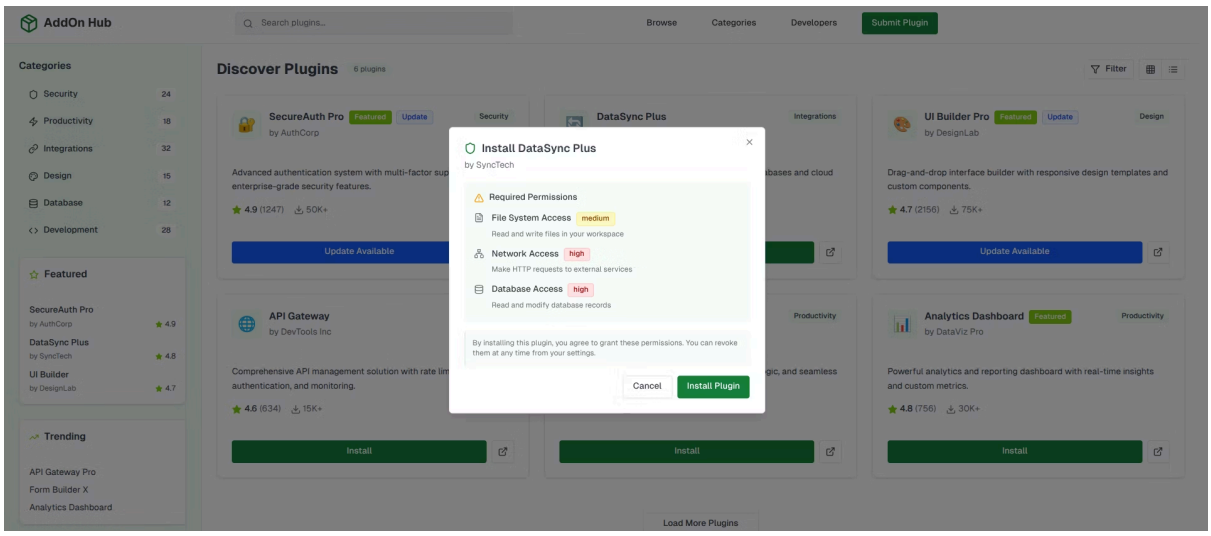
It also helps isolate the host web application from being affected if the plugin (accidentally) introduces a new vulnerability. When this sandbox is inadequately implemented or missing, it can allow attackers to bypass it and gain unauthorized access to the server, read sensitive data, or reach other authenticated parts of the application.

As a researcher, you'd want to develop a plugin with the aim of, depending on your target, extending your current in-app privileges. For instance, you could check if you can access other plugins' data, delete existing or install new plugins, grab session cookies, or perhaps even reach protected API endpoints that allow you to perform unwanted actions.

2. Improper access controls

Besides sandboxing, developers must also implement proper access controls to define what plugins can do inside their execution environment. In some instances, you'll notice that several add-ons have or request more permissions than necessary, or platforms fail to strictly enforce the principle of least privilege.

Suppose the platform allows you, as a plugin developer, to request access to perform certain actions on behalf of the user, for instance, to publish a public post. Try to reject this permission as the user while still attempting to publish a post with the plugin enabled.



Testing for overly permissive plugins

In addition to testing for vertical access escalation vulnerabilities, try also to include testing for horizontal access control issues, such as attempting to modify the settings of another plugin.

3. Overly-permissive APIs

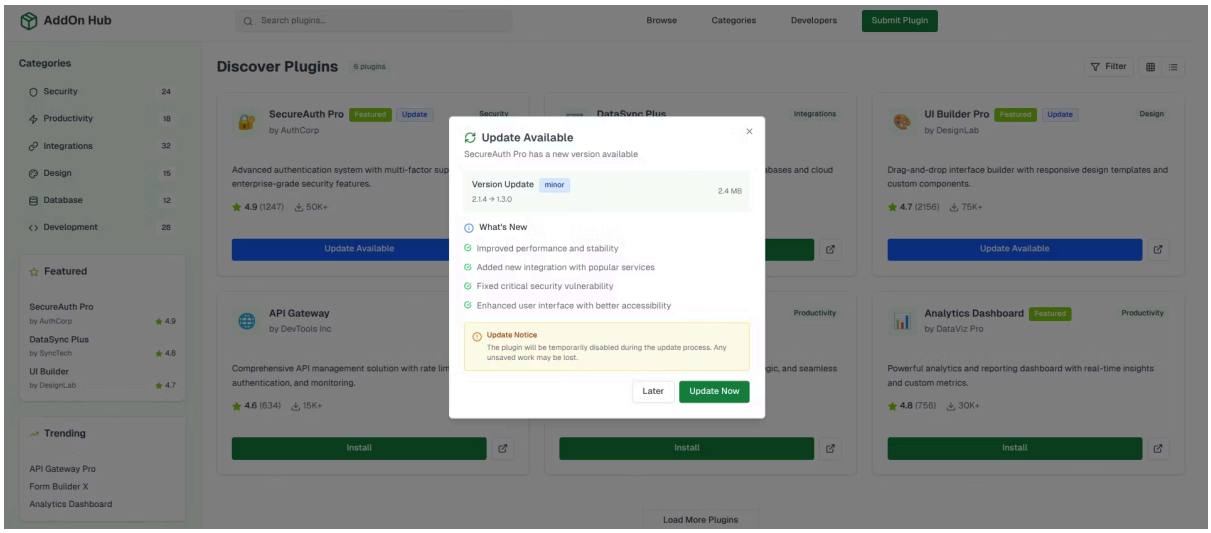
Platforms that feature a plugin ecosystem often also expose an additional API to plugin developers. Usually, this API won't receive the same security attention as the main API will. This opens up another attack surface for us to test for lack of rate limiting, authentication, and authorization vulnerabilities.

If plugins use the same API as the main application, try creating a plugin that attempts restricted actions. For instance, build a plugin that tries to reset a user's password or change their email address. Normally, these actions require additional verification, like 2-FA. But sometimes platforms treat API requests from plugins as 'trusted' and skip these security checks. This means your plugin might be able to perform actions that would trigger security checks through the normal interface. Test this by having your plugin make API calls to endpoints behind 2-FA security checks and see if the platform properly enforces the same security restrictions as it would for a regular user request.

4. Supply chain attacks

Cloud-based plugin ecosystems can also be susceptible to supply chain attacks during plugin installation or future updates. It's therefore recommended to include testing for this vulnerability type. For instance,

consider how plugins are installed and (automatically) updated. And if there are any strict background vetting or code review processes going on before new plugins are published to the public.



Supply chain attacks in add-on/plugin ecosystems

When no security processes are present, it can leave an open spot for us to publish plugins with malicious code. Compromising other popular plugins (for example, through an authentication or any other authorization vulnerability) could put a significant percentage of the user base at risk.

Even if the platform is cloud-based, examine how new updates are pulled and installed. And look for weak spots that could allow you to tamper verified plugins. When testing for such vulnerabilities, it is recommended to adhere to the program rules at all times by using test accounts whenever possible so that you can limit any potential impact.

Conclusion

Add-on/plugin ecosystems can help introduce lots of new use cases to the platform. However, when security is not prioritized and when best practices are not followed, it can open up a new, significant attack surface that we can take advantage of.

So, you've just learned how to test for vulnerabilities in plugin ecosystems... Right now, it's time to put your skills to the test! You can start by practising on vulnerable labs and CTFs or... browse through our 70+ public bug bounty programs on Intigriti and who knows, maybe earn a bounty on your next submission!

START HACKING ON INTIGRITI TODAY