# Proof of Concept

If we reach this point, it means that we have confirmed the existence of a vulnerability in the application/function we are testing. In this step, we would seek to understand how to reach the highest possible exploitation impact on the target by testing everything step by step. Then, we can write an exploit to automate all of that. Finally, we can slightly modify our script to target the real production server. If the exploit causes downtime or data loss, testing it on the real target is not advised.

## Full Chain Exploitation

Our first step would be to document a working exploitation process step-by-step. We should already have an excellent idea of how this application can be exploited, as we confirmed this in the Local Testing step. However, now we would need to document every step and payload and will also need to bypass any restrictions that may hinder our exploitation process.

This may be pretty challenging since we 'in a way' took the easy route in testing, as our goal was simply to confirm the existence of the vulnerability. On the other hand, actual exploitation may be more challenging, especially if the exploitation process chained multiple vulnerabilities or if the application had a lot of protections in place (e.g. WAF or Anti-Virus). Still, with advanced knowledge of this vulnerability and its potential bypasses, we should be able to achieve full exploitation.

This varies from one pentest to another since every vulnerability is different. Generally, however, we need to note down the following:

1. Initial target location
2. Client-side payload
3. Any other payloads (when chaining multiple vulnerabilities)
4. Potential/working bypasses (e.g. in payloads or web requests)

We also often follow a similar path as we did during our local testing, so we would start from the client-side user-controllable input and document what payloads would be needed to complete the vulnerability exploitation. Suppose we had to chain multiple vulnerabilities to exploit the final intended target (e.g. one to bypass authentication and another to reach remote code execution). In that case, we must note each step and payload for each. If, at any stage, a standard payload is rendered invalid due to a security mechanism (e.g. WAF), then we would start working on bypassing it before moving to the next step. Once we achieve a full chain exploitation (from client-side), we can begin developing a script that automates each step.

If we could not achieve full exploitation on the real target, we can still report the vulnerability. However, the vulnerability's severity may be reduced, leading to a lower success or bounty rate.

**Tip:** If you ever do identify a vulnerability, then try to persist until you can achieve exploitation and bypass all security mechanisms in place. If you don't, then someone else will likely be able to exploit your findings through the use of better bypasses or chaining other vulnerabilities.

## Exploit Development

This step is mainly about automation, where we write a script that automatically reproduces the steps we detailed above. The language we use for the script depends on many factors, but here are some guidelines:

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Recommended Language** | **Reason** |
| Attack is on a network application (including web applications) | Python | It works similarly on most operating systems |
| Attacking a client-side function (e.g. a CSRF attack) | JavaScript | It is the only script executed by browsers |
| Web chain including a client-side attack | Python & JavaScript | We prepare a JavaScript payload for the client-side part. Then, use it with a Python script to trigger the exploit and carry on the rest of the back-end attacks. |
| Binary exploitation | Python | Python has good libraries for debugging and exploiting binaries, while C/C++ may be used to develop a binary exploit. |
| Targeting an operating system | Bash or PowerShell/CMD | Whatever pre-installed scripting language on that operating system |
| Thick client or some advanced types of exploitation | The application's programming language | This would enable us to reuse code/functions to generate some payloads, which would save us a lot of time (vs re-scripting all of the logic in Python) |

Once we select a language, we can start developing a script that automatically exploits the target. We will not go through any exploitation details in this section, but it will be thoroughly covered later in the module and throughout other Academy modules. Each module following a whitebox approach will provide a different example of exploit development, and will demonstrate various tips and techniques we can utilise when developing exploits.

## Test on the Real Target

The final (and easiest) part is to test our exploit on the real target. We can do so by modifying the target details in the script (e.g. target's IP address and Port). We should only test our exploit after thoroughly testing it against the test target (with all security mechanisms enabled) and ensuring it works fully automatically and safely.

We should make our exploitation process fully revertible, as well as make it automatically clean any traces of the exploitation process. The following are a few things we should keep in mind:

1. If we needed to create a new account, we would need to delete it afterwards (if possible)
2. If we modify any data (e.g. reset an admin password), we would make sure to reset it afterwards
3. If we ever interact with the back-end server OS or with the DB, we need to make sure to clean any traces we leave
4. We should never modify critical data and always test on data we create or trivial data
5. As the exploit may fail midway through execution, we should handle errors in a way that it will revert whatever was executed so far, even if the exploit did not succeed
6. We must always ensure that any tests we carry do not lead to any downtime, data loss, or permanent data modification.

Once our PoC successfully runs on the production target, we may move on to the final Patching and Remediation step.