Enumeration of errors and stack traces

Introduction

Stack traces and errors often contain valuable information about the targets we are testing. They often expose some internal workings of the websites we testing and might give us an idea on what we need to do to get our attack scenario to work. Things we might find in stack traces and errors can include:

- Dependencies being used
- Some source code being used
- Implementations of programming principles
- What values you are allowed to enter
- Why your value was wrong
- If the error says "Account not found" and "password incorrect" it is way to verbose
- Version numbers
- Memory dumps
- Ip addresses from the internal network of the server

Test objective

Besides the obvious objectives of "Identify existing error output" and "Identify existing error output" as defined by OWASP, we want to make sure we understand every parameter the application uses and what kind of value is expected. We want to identify the unexpected inputs (such as too much characters or a string instead of an integer) and we want to make sure we try them all. The more "Rainy day scenario's" (Scenario's which the application does not expect) we can think of, the better.

You may have noticed by now that errors and stack traces will seldom gather us a direct attack vector but the information contained in them can be of vital importance.

Make sure you also test for reflected XSS if the page contains a reflection of the tampered parameters. (For example 404 - Page /<img src=x onerror=alert()> not found if we visit the URL "localhost/<img src=x onerror=alert()>" which might pop an XSS).

Testing for these errors and stack traces

We are going to make a distinction between web servers and websites as both can generate errors which can manifest in different ways.

How to test web servers

- Test 404 pages
- Test 403 pages (tip: .htaccess might trigger 403 even if it does not exist)
- Requests that breaks the HTTP RFC

How to test websites

- Find all the user controlled parameters
- Find out what input is expected. Pay attention to number of characters, data type, negative numbers, incorrect formatting on things like email,...
- Execute all unexpected scenario's that the tester can think of, preferably based on requirements coverage if the requirements are available to avoid spending infinite time testing everything

- Understand any errors that are being generated and see if you can manipulate the input to generate more errors
- On testing environments debugging logging might be enabled to more easily trace any issues. This has to be disabled on production environments.

Error handling and what to log when

Error handling is a sensitive topic since the developers want as much information as possible to debug any issues that occur but we don't want to give any attacker information about our system architecture. For this reason it's important that we log the correct things at the correct priviledge levels.

Internal systems that can not be viewed from the internet can contain a very detailed logging line per defect that occurs but we need to ensure these errors never reach the end user. The end user needs to only see a generic error message to ensure they do not get a glimpse into the internal workings of our application.

To achieve this we can create an error handling class which will trigger at the moment any error occurs and which will translate the error message or stack trace into a human readable generic text. Furthermore logging frameworks should be used and secured on all levels. This includes the storage of the logs and the logging framework itself.

Extra resources

- https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html
- Any 403 error messages give away that a resource exists. Preferably show a generic error message to the user
- https://www.veracode.com/security/error-handling-flaws-information-and-how-fix-tutorial
- https://owasp.org/www-community/Improper_Error_Handling
- https://pentestbook.six2dez.com/others/web-checklist#error-handling