Team Too: OWASP EnDe

Joshua Lusk, Samuel “Ethan” Price, Lauren Donner, Zachary Kidd

Introduction to EnDe

EnDe is a project under OWASP that provides an encoder, decoder, converter, transformer, and calculator. It aims to meet the requirements for HTTP/HTML-based functionality. EnDe only requires that you have a web browser installed and a single folder to install files into. It is run from its “index.html” file.
Since there is no single way to encode, decode, convert, or translate any one thing, the project itself is coded in several languages, and is run by HTML. The file “index.html” pulls from the resources in the containing folder to run all the functions this project intends to use, and runs simply in a web browser. This makes the project widely available to everyone intending to use it regardless of which language development kits they have installed.

EnDe GUI

Architecture

/TestAutomation
- /project
  - /src
  - /bin
- /scripts
  - runAllTests.py
- /testCases
  - testCase1.txt
  - testCase2.txt
- /testCaseExecutables
- /temp
- /oracles
- /docs
  - README.txt
- /framework
- /reports

Test Case Files

12 lines (10 sloc) | 344 Bytes

```
test_number: 12
requirement_being_tested: Convert plain text to Atbash encoding
component_being_tested: Atbash encoding function in EnDe.js

driver: qunit_driver
        language: javascript
        method_being_tested: atbash(src)
        test_input: src = "This is plain text"
        expected_outcome: "Gsrh rh kozrm gvcg"
        method_invocation: EnDe.atbash(%s)
        arity: 1
```

12 lines (10 sloc) | 319 Bytes

```
test_number: 19
requirement_being_tested: Convert characters to hexidecimal
component_being_tested: the EnDe graphical interface - index.html

driver: selenium_driver
        language: python
        method_being_tested: NA
        test_input: ABCDXYZ
        expected_outcome: 4142434458595a
        list_number: 7
        link_id: EnDeDOM.EN.Actions.s.chrHEX
```

Test number, requirement being tested, component being tested, method being tested, test input, expected outcome, and language are common to both the JavaScript tests and Python (Selenium) tests.
For JavaScript tests, the files must also include the method invocation and arity.
JavaScript tests should also have test input and expected outcome in quotes.
For Python tests, the files must also include the list number (where the type of encoding, decoding, conversion, transformation, or calculation shows on the list in the GUI) and the link ID.
Python tests do not need to put test input and expected outcome in quotes, as they are read from the file as strings.

How It Works

- runAllTests.py reads through all files in testCases directory
- All values in each file in the testCases directory is added to a dictionary
- Each dictionary is added to a list, which contains all dictionaries for all test cases
- We no longer require files under the testCaseExecutables directory, all required information is in the test case files. This avoids repeated code in individual executable files.
- runAllTests.py will execute the method being tested based on the language noted in the file
- Python files test the GUI using Selenium to simulate mouse clicks/keyboard input
- JavaScript files test the functions directly
- JavaScript tests use QUnit, a JavaScript testing framework

Requirements

To run EnDe and our automated testing framework, the following hardware and software requirements must be met:

- Compatible Mac, Windows, or Linux operating system
- Mozilla Firefox
- Python (versions below 3.x)
- Selenium to run GUI tests

Fault Injection

We injected faults into the code, to make sure our tests failed when the incorrect output was given.

Test 10: Modified the reverse function in EnDe.js
Test 11: Modified the val2num function in EnDeCheck.js
Test 12: Modified the atbash function in EnDe.js
Test 15: Modified the DELwhite function in EnDeText.js
Test 18: Modified the Stibitz function in EnDe.js

All modified tests failed as expected.

We also created “add_faults.sh” and “remove_faults.sh”, which swap out our code with faulty code using the following commands:
./add_faults.sh
and
./remove_faults.sh

Noteworthy Experiences

- We ran into issues with Chrome, as it requires us to disable security features all together. Firefox needed to be used for our tests.
- Some methods could not be tested in the browser, as they return characters that are not recognized.
- Other methods that returned values that were not UTF-8 encoded broke the reporting (writing to results.html files), and had to be avoided. An example of a function that broke the reporting because of this issue was the AES-128 encoding function