Chapter 1 & Deliverable 1: OWASP EnDE

The Main Project

Since we've decided to start with OWASP, and due to the large number of projects within OWASP, we've selected a sub-project. This project is EnDe.

OWASP EnDe Project is a project that provides an encoder, a decoder, a converter, a transformer, calculator all at the click of a mouse. It contains a collection of functions for various codings, encodings, decodings and convertions used world wide. One of the aims of the project was to meet the requirements for HTTP/HTML-based functionality. Copy&paste must be possible, and some functions should be able to be called with a single click.

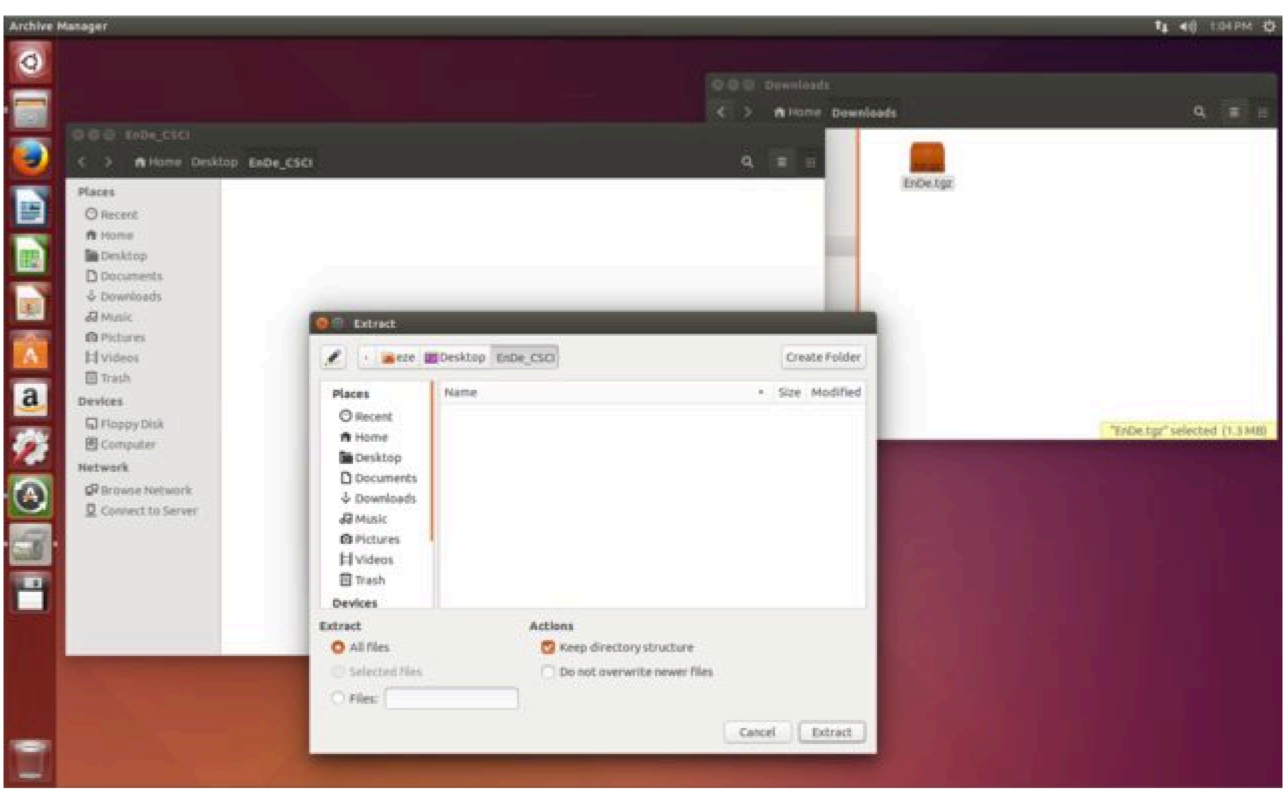
Since there is no single way to encode, decode, convert, etc.. any one thing, the project itself is coded in several languages, and is run by HTML. The file "index.html" pulls from the resources in the containing folder to run all the functions this project intends to use, and runs simply in a web browser, as to make the project widely available to everyone intending to use it regardless of which language development kits they have installed.

Installation

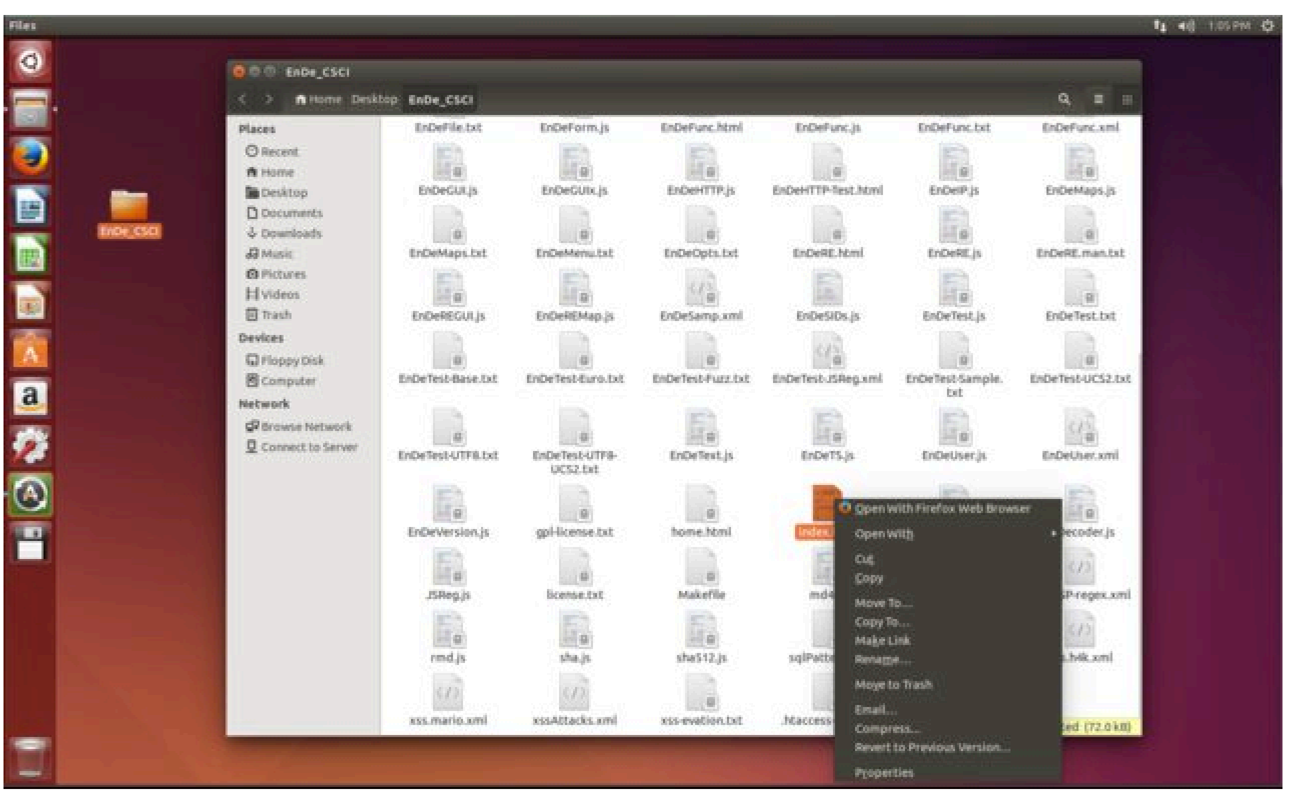
OWASP EnDE is made to be easy to use, so installation should be just as easy. The only requirements of installing the project is that you have a web-browser like Chrome or Firefox installed, and a single folder to install the files into. Once you have the files from the EnDe site downloaded, extract them and place them in the install folder. Open up EnDe by clicking on "index.html", and you are done.

(click on the pictures to open larger versions in a new tab)

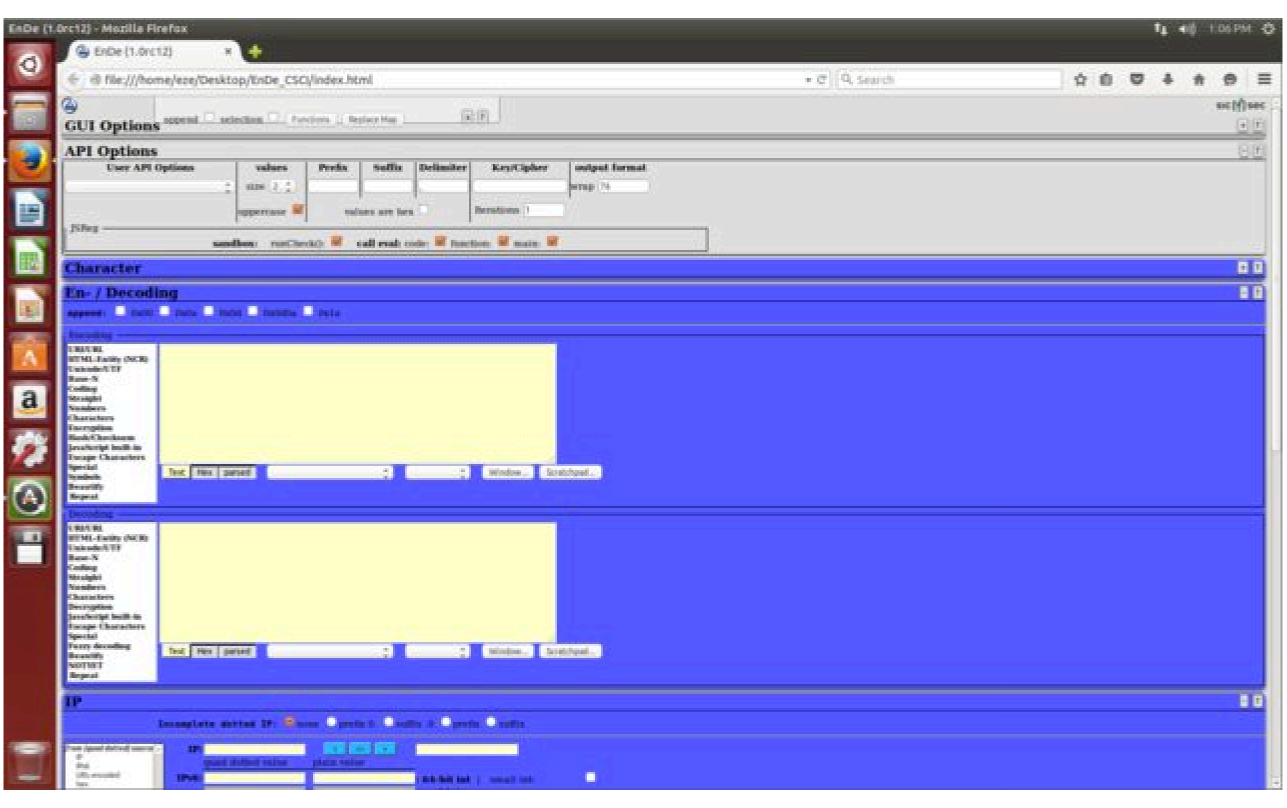
- Step1: download the zipped file needed, and extract it to an installation folder you've created.



- Step2: Open index.html, located in the installation folder, with your web browser.

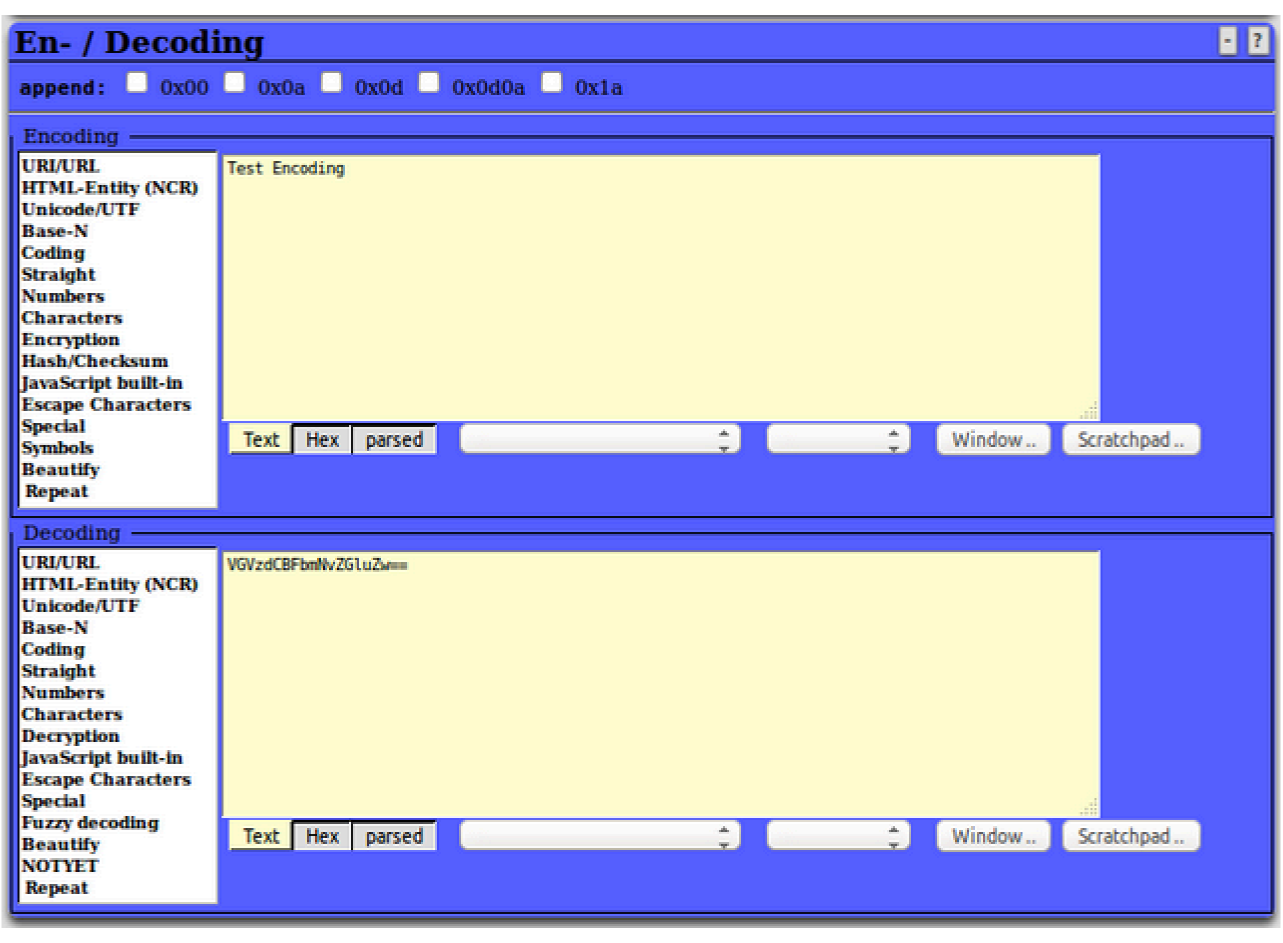


- Step3: Enjoy using the project EnDe!



Initial Testing

Simple encoding and decoding worked very well. Placing in a few of our own samples for Base64 encoding and decoding showed up exactly as they should have. The text "VGVzdCBFbmNvZGluZw==" decoded to "Test Encoding" and vice versa when encoding.


En- / Decoding

append: ☐ 0x00 ☐ 0x0a ☐ 0x0d ☐ 0x0d0a ☐ 0x1a

Encoding

URI/URL
HTML-Entity (NCR)
Unicode/UTF
Base-N
Coding
Straight
Numbers
Characters
Encryption
Hash/Checksum
JavaScript built-in
Escape Characters
Special
Symbols
Beautify
Repeat

Test Encoding

Decoding

URI/URL
HTML-Entity (NCR)
Unicode/UTF
Base-N
Coding
Straight
Numbers
Characters
Decryption
JavaScript built-in
Escape Characters
Special
Fuzzy decoding
Beautify
NOTYET
Repeat

VGVzdCBFbmNvZGluZw==

Existing Tests: Other examples with given results were used to ensure that the project was working correctly.

Simple Example 1 Using the string "Euro"

```
CRC-32 ------------- 1fb0ba15

base64 ------------- RXVybw==

urlUTF-8 ----------- Euro

MD5(hex) ----------- 3E823FAC7473E42888932C7761C224FC
```

Trying these encodings and hashes in various tools should always return the same result. Not very surprising, as it's expected to do so.

Simple Example 2 Replacing the 'E' in "Euro" with an '€' which results in "€uro"

```
CRC-32 ----------- b4c7b4a7

base64 ----------- IKx1cm8=

urlUTF-8 --------- %e2%82%acuro

MD5(hex) --------- 08c1d37a0b3394da278b77116cc4e615
```

Trying these in various tools will likely return different results depending on the tool used.