Using FUZZION

Intro

FUZZION is a command line program that is run with arguments to attack bWAPP. With some modifications, it can be made to attack other sites as well, though this would require more effort on the user's part.

Quick Start Guide

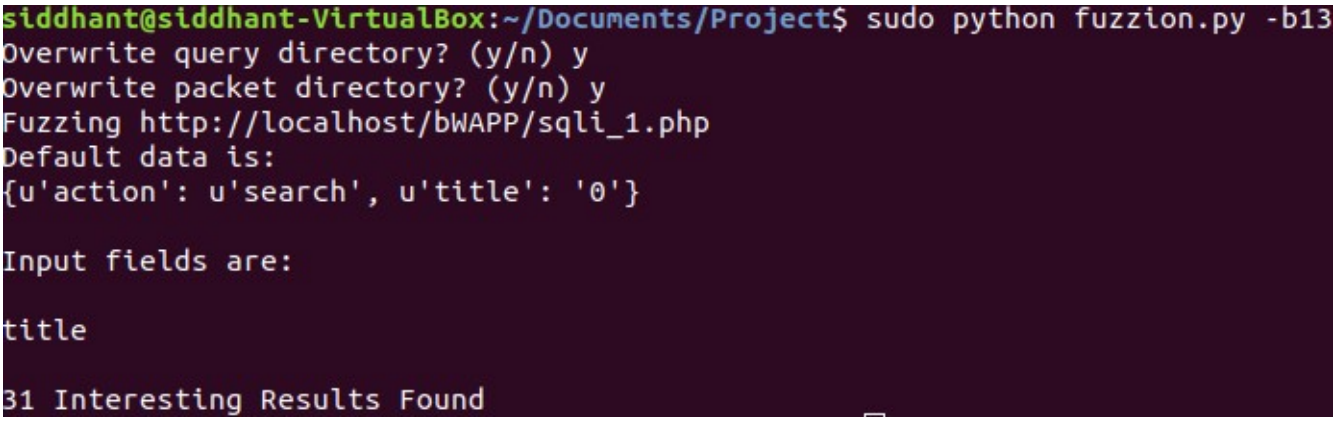
FUZZION is very easy to use right out of the box, and is run from the command line.

First, make sure that you have bWAPP located at http://localhost/bWAPP. If not, modify line 38 in fuzzion.py to point to where bWAPP is installed.

To run FUZZION, go to the directory that you've unpacked FUZZION, enter the command prompt, and enter

```
python fuzzion.py -b13
```

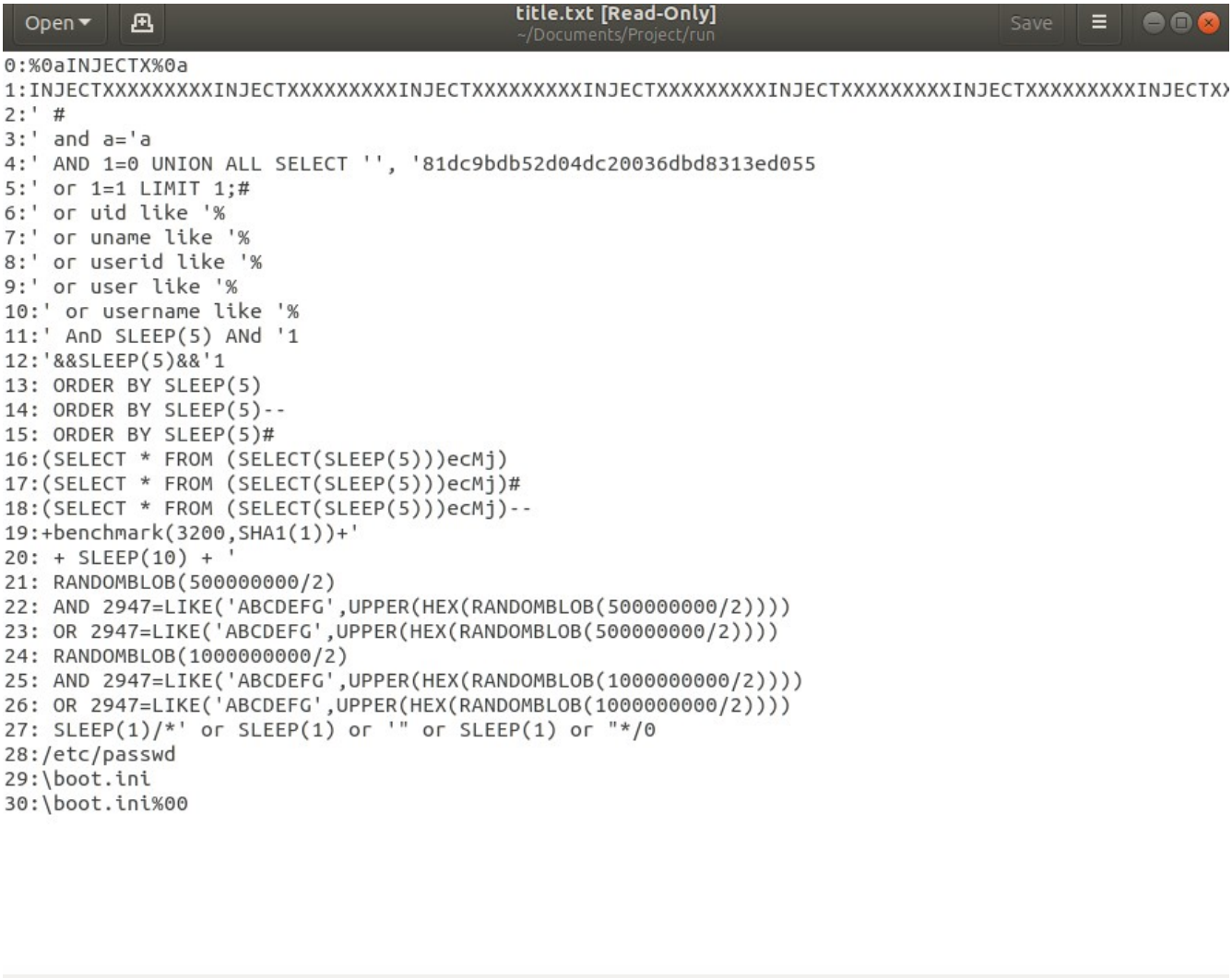
This will tell the program to attack bug #13 on the bWAPP site. Some sample terminal output is shown below:



Note that the execution states the input fields are "title". This means that the "title" field on the page is being fuzzed.

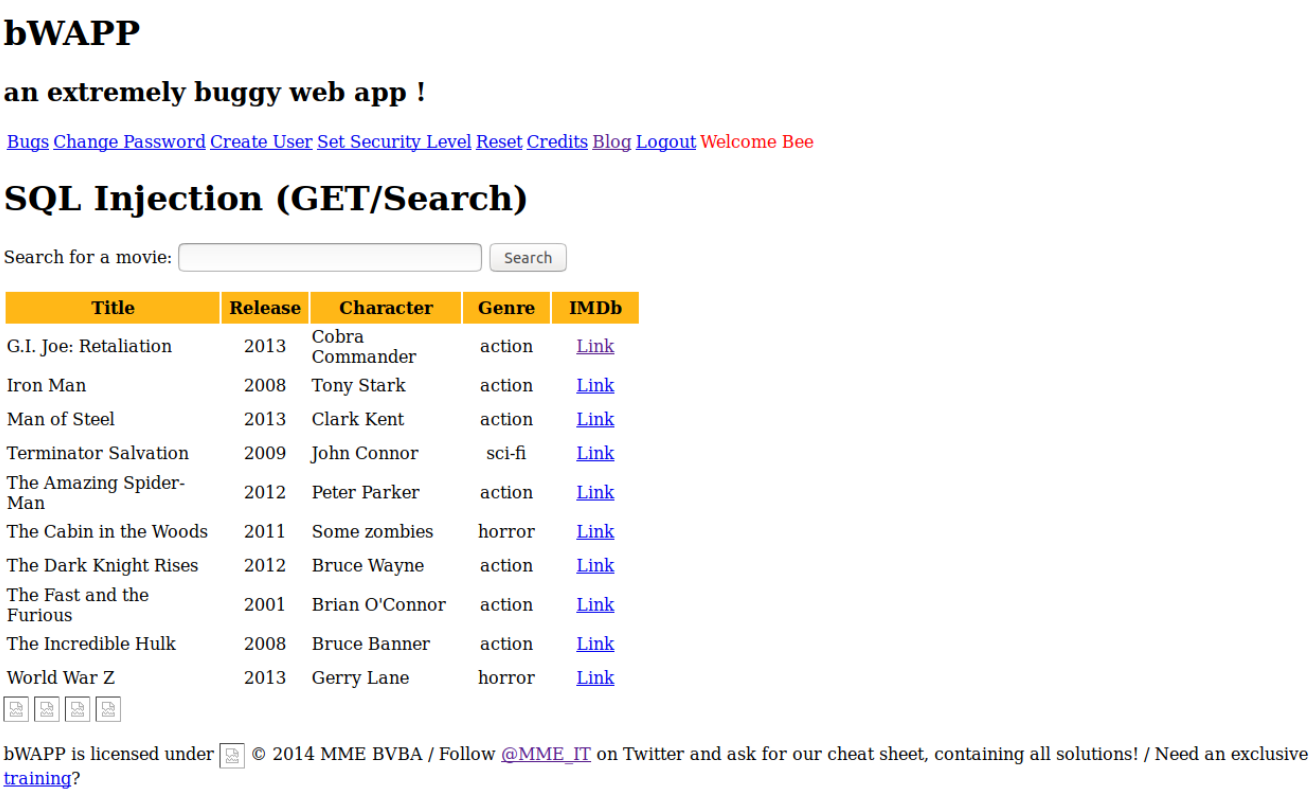
The example results are included in a folder called b13. Your results will be found in new folders under your working directory called "run" and "returned".

In the "run" folder, you'll find a file named "title.txt". This file corresponds to a fuzzed input. An example is shown below.



The file has lines in the format <number> : <input>. Going into the “returned” directory, you will see files named “<x>.html” where x is the number corresponding to fuzzing that field with the specified string.

Let’s look at 2.html by opening it with Mozilla Firefox.



bWAPP

an extremely buggy web app !

Bugs Change Password Create User Set Security Level Reset Credits Blog Logout Welcome Bee

SQL Injection (GET/Search)

Search for a movie: Search

Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Fast and the Furious	2001	Brian O'Connor	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link

bWAPP is licensed under © 2014 MME BVBA / Follow @MME_IT on Twitter and ask for our cheat sheet, containing all solutions! / Need an exclusive training?

Note how all results are listed for input string "' #". This is indicative of an exploit present in the system.

Other files in the "returned" folder are labeled "<x>.timeout", which means that the corresponding input string caused the server to time out. This is also indicative of an exploit.

Input Options

Options:

-h, --help

show this help message and exit

-b BUG, --bug=BUG

bug # on the bWAPP site (int)

-q QUERYDIR, --query-output=QUERYDIR

Where to store successful malicious queries

-r PAGEDIR, --page-output=PAGEDIR

Where to store returned pages

-u USER, --user=USER

preferred bWAPP username

-p PWD, --password=PWD

password for account

-s SEC, --security_level=SEC

bWAPP security_level

-f FUZZ, --fuzz_set=FUZZ

Set of fuzzing inputs

-i INP, --field_input=INP

File for manually set field inputs (format is

'field':'data')

-b IS A REQUIRED ARGUMENT. It is the bug on bWAPP that fuzzion will attack.

-p, -u, -s options are used for setting bWAPP login credentials and settings.

-f defaults to 0, which is a general fuzzing set. Can use -f1, -f2, etc to specify a different set. Details can be found in fuzzStrings.py. The actual fuzzing strings are located in the IntruderPayloads/fuzzLists directory.

-q, -r specifies where you want your successful fuzzing inputs and corresponding returned web pages stored, respectively.

-i is an input file that holds a field fixed while fuzzing other fields. Useful for password fuzzing as you can hold a username fixed while you fuzz passwords.

How it Works

FUZZION works by first logging into the bWAPP site. It then navigates to the desired attack page, and parses the main web form and url for potential inputs. It then begins sending HTTP requests that use fuzz strings for each field in turn. Each field is fuzzed separately.

The fuzz strings currently come from long text files full of input strings known to be problematic. These strings are obtained from fuzzStrings.py, which has several sets of fuzzing files that can be specified. The user can modify fuzzStrings.py to use their own files if they so choose.

The HTTP requests are sent, and the responses are collected. The responses are then parsed and thrown out if they contain error messages, echos of longer inputs, or produce pages that have already been seen.

These pages are finally saved to the packets output directory.

A More Complex Example

Let’s try fuzzing the php injection page (-b11), and for fun, at a higher security level. What you’ll notice is that there is no webform, or any fields parsed from the page either. However, clicking the message link redirects you to a url where there is a message field in the url.

Fuzzing this new page would yield results, as there is now an input field. However, we can instead note that the field is named ‘message’.

Let’s open a file named input.txt, and have it be just the following:

```
message:
```

This will be read in by FUZZION, and FUZZION will read it as an additional input parameter, choosing to submit fuzzing strings for it in addition to any other fields it finds on the page.

```
python fuzzion.py -b11 -i input.txt -f0 -s1
```

The -i flag will specify your input file. -f0 indicates it will use fuzzing set 0, which it already would have since 0 is the default. Add a -q or -r input if you want your output directories to be named something specific.

Here is the command line output.

```
siddhant@siddhant-VirtualBox:~/Documents/Project$ sudo python fuzzion.py -b11 -i
 input.txt -s1
Overwrite query directory? (y/n) y
Overwrite page directory? (y/n) y
Fuzzing http://localhost/bWAPP/phpi.php
Default data is:
{'message': '0'}

Input fields are:

message

39 Interesting Results Found
```

And the contents of “message.txt” under your -q directory (default “run”).

```
0:PUT
1:true
2:1
3:0
4:-1
5:2
6:3
7:4
8:5
9:6
10:7
11:8
12:9
13:INJECTXXXXXXXXXXINJECTXXXXXXXXXXINJECTXXXXXXXXXXINJECTXXXXXXXXXXINJECTXXXXXXXXXXINJECTXXXXXXXXXXINJECT)
14:==
15:'
16:' --
17:' #
18:'--
19:'/*
20:'#
21:" --
22:'\"
23:`
24:||
25:'"
26:"
27:"'
28:&
29:&&
30:%0a
31:dir
32:\n
33:pwd
34:\r\n
```

The directory “b1_s1” contains the example outputs for this already. Inspecting the html files that correspond, we find that the results are actually rather uninteresting. String #13 just returns a URL too long error. The rest of the interesting strings are less than 5 characters long, and the html pages just echo them.

This is because the echo check only excludes pages when the input strings are longer than 4 characters. Otherwise, the echo check can exclude pages that do actually show exploits.

Running the same command with -s0 instead will show many strings of actual interest, such as strings with periods having their periods removed. Perhaps the user can find a new set of fuzzing inputs, or generate their own to pierce medium security on the page.

Watching Packets with Wireshark

When you enter a fuzz string returned by FUZZION, you can observe the communication with wireshark. If you’re running the server locally, select loopback.lo

If you do OS COMMAND INJECTION (-b9) with fuzzing set 3, (-f3), then you can see, via the loopback interface, the server falling into a loop where it repeatedly tries to ping us.

List of Good Targets on bWAPP

- GET/SEARCH : -b13
- GET/SELECT: -b14
- POST/SEARCH: -b15
- POST/SELECT: -b16
- OS COMMAND INJECTION: -b9 -f3 (Will require server reboot)
- OS COMMAND INJECTION BLIND: -b10 -f3 (Will require server reboot)
- SESSION MANAGEMENT (ADMIN PORTAL): -b41
- XSS REFLECTED: -b49
- And more. The author has not run a script that loops through all -b values to examine the results for every single bug, due to lack of time. It can be done however, and is an exercise left to the reader.