Final Report: Less Pass

Team 1337

Soren, Alex, Holden, Griffin

Table of Contents

Introduction

LessPass is a free (libre) password manager that achieves higher security than competing password managers while being available to everyone for free. It achieves this security by being stateless. That is, rather than storing a user's passwords remotely, it generates the user's passwords based a three parameters: the website/service, the username/email, and a master password. The same master password can be used across all of a user's logins, exactly like other managers, because the three parameters are hashed to create each password. Other parameters can be passed to specify password length, caps, etc. It is a comprehensive project with a core library that serves web, web extension, mobile, and command line interfaces. It is made with node.js, but some of the interfaces use Django or React. We chose LessPass because it was not too mature in the development cycle and we knew we could improve on the test cases.

Chapter 1: Building the Project

The following steps include how we initially built the project. We used the Linux system agnostic build guide, other distributions have more complicated processes but this one is universal.

1. mkdir build && cd build
   - Makes a directory called build and enters it

2. cmake -DUNIX_STRUCTURE=0
-DCMAKE_INSTALL_PREFIX="${HOME}/obs-studio-portable" ..

   - Sets the environment variables to install in a specific directory, namely ~/obs-studio-portable

3. make -j4 && make install

   - Run 4 make jobs in parallel and make install once they complete

4. finally, go to '~/obs-studio-portable/bin/64bit/' and run './obs'

Chapter 2: Test Plan

Luckily for us both the core and the interfaces have existing unit tests. However, in select areas, shortcomings can be found with the existing tests. Therefore the plan is to modify bad tests as well as add more comprehensive tests. Below are some test case examples.

Clipboard

This is a test of the CLI, which simply takes in the parameters and gives an output. There are a few exceptions such as cases like this one:

Figure 1

```
test("options can be before parameters", async t => {
  const { stdout } = await execa("./cli.js", [
    "-C",
    "lesspass.com",
    "contact@lesspass.com",
    "password"
  ]);
  t.is(stdout, "Copied to clipboard");
});
```

It is possible to pass a parameter to put the generated password directly onto the clipboard, but all this tests is that "Copied to clipboard" was printed. Clipboards need very extensive testing, because each operating system implements clipboards in different ways, making the tests system dependant. Another reason testing the clipboard is difficult is because it can't be isolated. e.g. how could a test account for a user copying something while the test runs in the background? There are several tests involving the clipboard, and they most likely all need complete rewrites. Also, they probably need to be rewritten for each operating system. Sending the password directly to the clipboard is important because the only alternative on the CLI is printing it, which is obviously problematic.

Password rendering

For LessPass, making a password is a 2-step process. First, the parameters are encrypted and hashed. Then, the result of the first step is used to make a password that holds up to the options given by the user. This is called password rendering. The password rendering file only has a few tests, and only tests on one set of parameters.

Chapter 3: Getting Ready for Tests and Running the File

(Refer to figure 2 for commands used in each step)

Start by cloning the LessPass repo(1). Then Install Node on Ubuntu(2). Now JavaScript files can be run(3). In "lesspass/cli" there is a script, "cli.js" and a test file "test.js". We had to install Clipboardy to run "cli.js"(4). The script can then be run, but it takes options. Try "--help" to see if it works(5). See the readme on the cli page (https://github.com/lesspass/lesspass/tree/master/cli) for more examples. It doesn't need to be installed though because we have the script cloned. Replace the command "lesspass" with "./cli.js" to see that they are the same onto the test file. It's not possible to just run "node test.js" because the test file uses a testing framework called ava (https://github.com/avajs/ava) so we needed to install ava(6). After that we can run the test file(7).

Figure 2

```
1. git clone https://github.com/lesspass/lesspass.git
2. sudo apt install nodejs
3. node <filename>
4. npm install clipboardy
5. ./cli.js --help
6. npx create-ava --next
7. npm test
```

Chapter 4: Testing Framework and Cases

How our Framework Works

Each test case is executed with one command(shown below). Regex expressions used as Oracles and the outputs of the cases are tested against them for matches.

List of Test Cases

1. Tests default values (-luds --length 16 --counter 1)
   -luds: allow all character types
   --length 16: password length 16
   --counter 1: counter of 1
2. testing lowercase only (-l)
   example: cnduxoiwpnrufjk
   default length (16) and default counter (1)
3. testing uppercase only (-u)
   example: CNDUXOIWPNRUFJK
   default length (16) and default counter (1)
4. testing digits only (-d)
   example: 194875937485738
   default length (16) and default counter (1)
5. testing symbols only (-s)
   example: #$%^&*()[]{};'|
   default length (16) and default counter (1)
6. testing lowercase and digits only (-ld)
   example: 5nd8xoi2p7r17j6
   default length (16) and default counter (1)
7. testing lowercase and digits only (-ld)
   example: 5nd8xoi2p7r17j6
   default length (16) and default counter (1)
8. testing lowercase and symbols only (-ls)
   example: %n){xoi^p/r#-j.
   default length (16) and default counter (1)
9. testing uppercase and digits only (-ud)
   example: C55U281W2N7UF0K
   default length (16) and default counter (1)

10. testing uppercase and symbols only (-us)
    example: C&*U_=$W!N;UF,K
    default length (16) and default counter (1)
11. testing digits and symbols only (-ds)
    example: 8&*2_=$4!0;17,9
    default length (16) and default counter (1)
12. testing uppercase and digits and symbols only (-uds)
    example: C&5U9=$2!N;4F0K
    default length (16) and default counter (1)
13. testing lowercase and uppercase and digits only (-lud)
    example: Cj5U9dn2sNq4F0K
    default length (16) and default counter (1)
14. testing lowercase and digits and symbols only (-lds)
    example: (j5*9dn2s$q4;0"
    default length (16) and default counter (1)
15. testing lowercase and uppercase and symbols only (-lus)
    example: (jU*PdnCs$qM;H"
    default length (16) and default counter (1)
16. testing lowercase and uppercase and symbols and digits (-luds)
    example: 4j.8P6-C23qM;5"
    default length (16) and default counter (1)
17. testing an invalid input for length (-l -1)
    example: 4j.8P6-C23qM;5"
    invalid inputs for length should cause the length
    flag to be ignored
18. testing an input for length (-l -20)
    example: 4j.8P6-C23qM;5"b7
    defaults are used for characters and counts (-luds -c 1)
19. testing an input for length (-l -20)
    example: 4j.8P6-C23qM;5"b7
    defaults are used for characters and counts (-luds -c 1)
20. tests different inputs for counter (-c) produce different results
21. testing no lowercase (--no-lowercase)
    example: C&5U9=$2!N;4F0K
    default length (16) and default counter (1)
22. testing no uppercase (--no-uppercase)
    example: (j5*9dn2s$q4;0"
    default length (16) and default counter (1)

23. testing no symbols (--no-symbols)
    example: Cj5U9dn2sNq4F0K
    default length (16) and default counter (1)
24. testing no digits (--no-digits)
    example: (jU*PdnCs$qM;H"
    default length (16) and default counter (1)
25. tests that a correct output was copied to the clipboard

Chapter 5: What we Changed/ What Failed

The file that our tests are based on is found in the LessPass repo under project/packages/lesspass-cli/index.js . This file can be run as-is with node.js to evaluate its functionality. The following changes were implemented and their effects documented as shown below.

1): Line 123

```
counter: cli.flags.counter || 1,
```

If the objects on either side of the OR operator are swapped,

```
counter: 1 || cli.flags.counter,
```

the short circuit logic is broken, and the counter value will default to a value of 1 ignoring the users inputted value. This change alters the result of test case #20

2): Line 60

```
l: { type: "boolean" },
```

If the type is changed to a string value

```
l: { type: "string" },
```

Test case #2 is broken

3): Line 83

```
console.log(generatedPassword);
```

If the master password is printed as shown here :

```
console.log(masterPassword);
```

All of the test cases break except for #25

4): Line 90

```
 return !["l", "u", "d", "s"].some(
```

All of the strings are replaced with the letter “q”

```
return !["q", "q", "q", "q"].some(
```

This breaks test cases #2-15

5): Line 124

```
length: cli.flags.length || 16,
```

If the default length of the password is changed to

```
length: cli.flags.length || 10,
```

This breaks all test cases but #19 because it is the only test case to not use the default length value