Information Systems Security Project 2

Akhil Pragallapati
Siva Sravana Kashyap Pathiki

Introduction:

Systematic testing helps us to cover classes of equivalent test cases. The testing effort is significantly decreased by specifying such test classes while maintaining test coverage.

This method has the disadvantage that we only test the things we anticipate breaking. This might enable bugs brought on by unexpected input data or side effects to pass the tests. and appear in systems used for production.

A method known as random testing involves automatically performing a large number of tests using randomized input data in order to enhance the coverage of the domain of our software's inputs. This could be entirely random "byte noise," largely reliable information delivered by a skillfully designed generator, or anything in between.

Fuzzing is a technique used in software testing to find flaws and vulnerabilities by feeding the system with erroneous or unexpected data. This can be accomplished using the fuzzing package that Python offers. In this report, we will use the Python fuzzing package's fuzz_string function.

For creating and altering data for fuzzing, the Python fuzzing module offers a number of functions. "**pip install fuzzing**" can be used to install this package. After installation, "**import fuzzing**" in Python can be used to import the package. The package includes a number of modules, including FuzzExecutor, fuzz_string, and generators.

Working:

Based on a seed, the "*fuzz_string*" function can be used to create a list of randomly produced strings. Three parameters are taken into account by the function: "*seed", "number_of_fuzzed_variants_to_generate", and "fuzz_factor"*. A string that serves as the starting point for creating new strings is known as the seed parameter. The amount of fuzzed variants to generate is specified by the "*number_of_fuzzed_variants_to_generate*" argument. The "f*uzz_factor*" option regulates the degree of fuzzing.

Based on the seed parameter that is the input, the "*fuzz_string*" function creates a list of randomly produced strings. The "*number_of_fuzzed_variants_to_generate*" argument specifies how many fuzzed strings should be produced. By supplementing the seed argument with random characters, the "*fuzz_factor"* parameter regulates the degree of fuzzing.

```
# set the number of iterations and the maximum length of the input
num_iterations = 1
max_input_length_int = 10
number_of_fuzzed_variants_to_generate = 10
fuzz_factor = 7
min_input_length_int_string = 30
max_input_length_int_string = 50
```

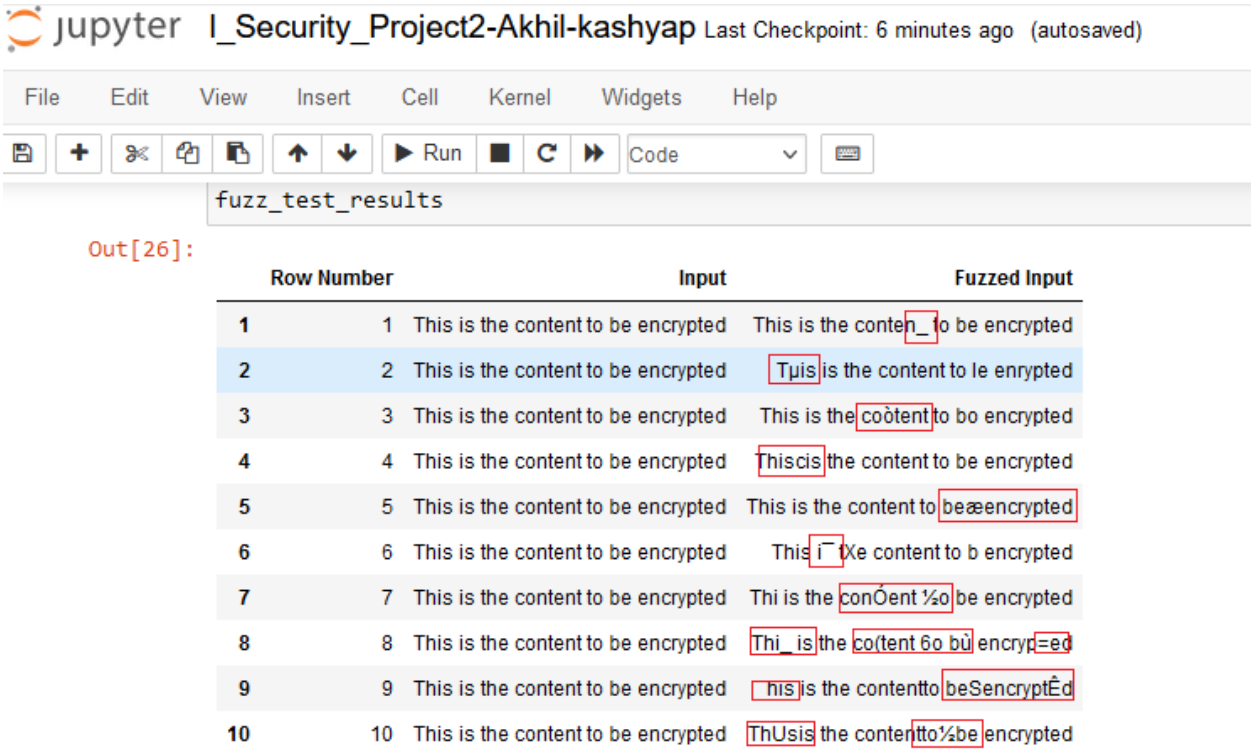
Information Systems Security Project 2

Akhil Pragallapati

Siva Sravana Kashyap Pathiki

We have implemented the **fuzzing** package with 10 variations (*number_of_fuzzed_variants_to_generate = 10*) for a given seed value. This seed value is a randomly generated string with minimum length of 30 ("*min_input_length_int_string = 30*") characters and a maximum length of 50 characters ("*max_input_length_int_string = 50*"). The degree of fuzzing is set to 7 ("*fuzz_factor = 7")*

For example:



	Row Number	Input	Fuzzed Input
1	1	This is the content to be encrypted	This is the conten_ to be encrypted
2	2	This is the content to be encrypted	Tµis is the content to le enrypted
3	3	This is the content to be encrypted	This is the coòtent to bo encrypted
4	4	This is the content to be encrypted	Thiscis the content to be encrypted
5	5	This is the content to be encrypted	This is the content to beæencrypted
6	6	This is the content to be encrypted	This i¯ tXe content to b encrypted
7	7	This is the content to be encrypted	Thi is the conÓent ½o be encrypted
8	8	This is the content to be encrypted	Thi_ is the co(tent 6o bù encryp=ed
9	9	This is the content to be encrypted	his is the contentto beSencryptÊd
10	10	This is the content to be encrypted	ThUsis the contentto½be encrypted

This fuzzed input is fed as an input to the "Alice Code" that was created for Encryption in Project 1. Then the encrypted content is fed as input to the "Bob Code" that was created for Decryption in the Project 1. In Project1, we used to compare the data to be encrypted and decrypted output. But in Project 2 the "*fuzzed_input*" generated by **fuzzing package** is compared with the decrypted output to find out if the encryption and decryption functions created in Project 1 are working well on the fuzzed inputs.

For example:

Information Systems Security Project 2

Akhil Pragallapati
Siva Sravana Kashyap Pathiki

Out[39]:

	Row Number	Input	Fuzzed Input	Decrypted Fuzzed Input	Result
1	1	This is the content to be encrypted	Thi is the content to be encrypted	Thi is the content to be encrypted	SUCCESS
2	2	This is the content to be encrypted	This i6 theecontent to be ecrypted	This i6 theecontent to be ecrypted	SUCCESS
3	3	This is the content to be encrypted	TÃis is the content to be encrypted	TÃis is the content to be encrypted	SUCCESS
4	4	This is the content to be encrypted	This is the cntent to9be encrypted	This is the cntent to9be encrypted	SUCCESS
5	5	This is the content to be encrypted	This y the content to be encry.ted	This y the content to be encry.ted	SUCCESS
6	6	This is the content to be encrypted	This is the content t$ be ´ncrypted	This is the content t$ be ´ncrypted	SUCCESS
7	7	This is the content to be encrypted	Thzs is the conte5t tB be enrypteÏ	Thzs is the conte5t tB be enrypteÏ	SUCCESS
8	8	This is the content to be encrypted	This is the coftent¾to be encrþted	This is the coftent¾to be encrþted	SUCCESS
9	9	This is the content to be encrypted	This is the content to be encrpted	This is the content to be encrpted	SUCCESS
10	10	This is the content to be encrypted	Th9s is the content to be encryptÏd	Th9s is the content to be encryptÏd	SUCCESS

In order to test the encryption and decryption functions we have used **100** randomly generated texts and created **10** fuzzed texts for each randomly generated text by keeping the "*fuzz_factor*" constant which gives a total of 1000 inputs to the encryption and decryption functions. Now these 1000 inputs are encrypted and then decrypted. Now the "*fuzzed_input*" is compared with the "*decrypted_text"* to find out if the encryption and decryption have worked correctly on the fuzzed inputs.

For example:

Row umber	Random Input	Fuzzed Input	Encrypted Fuzzed Input
1	xThmjrNiBtjlRGuflSmAHFcNpyamkBxgpykwgFfj	xThmjrNiBtjlRGuflSmAHFcNpy7mkBxgpykwgFfj	b'gAAAAABkVD8nGKrEZBRFKsUFIc95jupe-MMnLYdOL_1g..
2	xThmjrNiBtjlRGuflSmAHFcNpyamkBxgpykwgFfj	xThmjrNiBtjlRGuflSmAHFcNpyam$BxgpykwgFfj	b'gAAAAABkVD8njbA9jOdOnp7dCWn7E66d2y4TOgaaog64..
3	xThmjrNiBtjlRGuflSmAHFcNpyamkBxgpykwgFfj	xThmjrNiBtjlRGuflSmßHFcNpyamkBxgpykwgFfj	b'gAAAAABkVD8nGHtdXliIRSm8hithIg1JzwJS4ZZzesP0..
4	xThmjrNiBtjlRGuflSmAHFcNpyamkBxgpykwgFfj	xThmjrNiBtjlRGuf)SmAHFcNpyamkBxgpykwgFfj	b'gAAAAABkVD8nytiyWyYpJVXUJHjToj0LNHmpAap_utmw..
5	xThmjrNiBtjlRGuflSmAHFcNpyamkBxgpykwgFfj	xThmjrNiBtjlRGuflSmAH%cNpyamkBxgpykwgFfj	b'gAAAAABkVD8n55chafL_6ZkL2wmHa4sDb4j5ILZFvjK9..
...	...	...	..
996	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzAaLyTkfJzFbFG	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzØaLyTkfJzFbFG	b'gAAAAABkVD8vxzxysNd3YQSsEBOAOh0V26nLFiPjLPMH..
997	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzAaLyTkfJzFbFG	pjLSGSFwUHQzWVTMoiiFgVobClTmjzAaLyTkfJzFbFG	b'gAAAAABkVD8vFkPPPr8D8ukWHhzFLS3kNIaknKv_5D21..
998	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzAaLyTkfJzFbFG	pjLSGSFwUHQzWVTMopiFgVobClTmjtzAaLyTkfJzFbFG	b'gAAAAABkVD8v-c2iZRTMrfMp8CwZndI5b_6HLLWlzdlb..
999	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzAaLyTkfJzFbFG	pjLSGSFwUHQzWVTM}iiFgVobClTmjtzAaLyTkfJzFbFG	b'gAAAAABkVD8v_vm9xz0LHHaKdY6oErVQ-9i8gUjmpuic..
1000	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzAaLyTkfJzFbFG	pjLSGSFwUHQzWVTMoiiFgVobClTmjtzAaLyTkf©zFbFG	b'gAAAAABkVD8v1gHyKTO1DdsOcd_ohtcepXhkc5E8QmHy..

Encrypted Fuzzed Input	Decrypted Fuzzed Input	Result
b'gAAAAABkVD8nGKrEZBRFKsUFIc95jupe-MMnLYdOL_1g...	xThmjrNiBtjlRGuflSmAHFcNpy7mkBxgpykwgFfj	SUCCESS
b'gAAAAABkVD8njbA9jOdOnp7dCWn7E66d2y4TOgaaog64...	xThmjrNiBtjlRGuflSmAHFcNpyam$BxgpykwgFfj	SUCCESS
b'gAAAAABkVD8nGHtdXliIRSm8hithIg1JzwJS4ZZzesP0...	xThmjrNiBtjlRGuflSmßHFcNpyamkBxgpykwgFfj	SUCCESS
b'gAAAAABkVD8nytiyWyYpJVXUJHjToj0LNHmpAap_utmw...	xThmjrNiBtjlRGuf)SmAHFcNpyamkBxgpykwgFfj	SUCCESS
b'gAAAAABkVD8n55chafL_6ZkL2wmHa4sDb4j5ILZFvjK9...	xThmjrNiBtjlRGuflSmAH%cNpyamkBxgpykwgFfj	SUCCESS
...	...	...
b'gAAAAABkVD8vxzxysNd3YQSsEBOAOh0V26nLFiPjLPMH...	pjLSGSFwUHQzWVTMoiiFgVobCITmjtzØaLyTkfJzFbFG	SUCCESS
b'gAAAAABkVD8vFkPPPr8D8ukWHhzFLS3kNIaknKv_5D21...	pjLSGSFwUHQzWVTMoiiFgVobCITmjzAaLyTkfJzFbFG	SUCCESS
b'gAAAAABkVD8v-c2iZRTMrfMp8CwZndI5b_6HLLWlzdIb...	pjLSGSFwUHQzWVTMopiFgVobCITmjtzAaLyTkfJzFbFG	SUCCESS
b'gAAAAABkVD8v_vm9xz0LHHaKdY6oErVQ-9i8gUjmpuic...	pjLSGSFwUHQzWVTM}iiFgVobCITmjtzAaLyTkfJzFbFG	SUCCESS
b'gAAAAABkVD8v1gHyKTO1DdsOcd_ohtcepXhkc5E8QmHy...	pjLSGSFwUHQzWVTMoiiFgVobCITmjtzAaLyTkf©zFbFG	SUCCESS

Now, to test the encryption and decryption functions further, we have increased the fuzzing complexity by increasing the "*fuzz_factor*". The "*fuzz_factor*" is increased from 10 to 19 consecutively and as mentioned above, for each value of "*fuzz_factor*", 1000 fuzzed inputs are generated. This brings the total number of fuzzed inputs to the encryption and decryption functions to 10000 (i.e., 1000 for each "*fuzz_factor*" value ranging from 10 to 19).

For example:

```
fuzz_test_results
```

Out[43]:

	Row Number	Random Input	Fuzzed Input	Encrypted Fuzzed Input	
1	1	vlyBmEKsiwuMfZPZZSmjalLfObWMhRdyhb	vlyBmEUsJwuMfZPZZSmjalLfObWMhRdyhb	b'gAAAAABkVEes1C-Qj_Btl6iXPArDEM_3_6fp9Tx_EqUV...	vlyBml
2	2	vlyBmEKsiwuMfZPZZSmjalLfObWMhRdyhb	vlyBmEKsi{uMfPZZSmjalLfObAM%Rdyhb	b'gAAAAABkVEeszliysDdOZHbYwxldh2d0P0ljqMoYPmn8...	vlyE
3	3	vlyBmEKsiwuMfZPZZSmjalLfObWMhRdyhb	vlyNmEKsiwuMfZPZZSmjalLfObWMhRdyhb	b'gAAAAABkVEesvJ7SXiDwQGdHLkJAUNJUwnYu9fEe3gcJ...	vlyNm
4	4	vlyBmEKsiwuMfZPZZSmjalLfObWMhRdyhb	vlyBmEKsiwuMfZPZZSmjalLfObWMhIdyhb	b'gAAAAABkVEesUu0XviVVJwPVBaDC0f6vXxpCLMfkGUHg...	vlyBr
5	5	vlyBmEKsiwuMfZPZZSmjalLfObWMhRdyhb	vlyBm-KsiwuMfZPAZSmjalLfObWMhRdyh	b'gAAAAABkVEesQTxLQUy4Qp9coKAHvHHupi0rOn0d-UWe...	vlyE
...	...	...	...	...	
9996	9996	SyoESdzfeJsGhCPYLTizsKhjgovklWFfC	SyoESdzfeJsGhCPYLTizsKhjyovèlWFfC	b'gAAAAABkVEgK89rrDlcWuPzONPEYOxOz0nHmkSqZ960Z...	S
9997	9997	SyoESdzfeJsGhCPYLTizsKhjgovklWFfC	SyoESdzfeJsGhCPÉLTézsKhjgovklWFfC	b'gAAAAABkVEgKKnsPzDYyNrr-zTQpY_bE7F-O6ZbfJX_o...	Sy
9998	9998	SyoESdzfeJsGhCPYLTizsKhjgovklWFfC	SyoESdzfeJsG0CPYLTizsKhjgovklWFfC	b'gAAAAABkVEgK82VicKP_H8pWr_1zyVqRMfqq2UTvRwHx...	Sy
9999	9999	SyoESdzfeJsGhCPYLTizsKhjgovklWFfC	SyoES4zfeJsGhCPYLTizsKhjgovklWFf	b'gAAAAABkVEgKFzmUxc4GnvlAh-DK6GujSgyNMUyd2zh_...	
10000	10000	SyoESdzfeJsGhCPYLTizsKhjgovklWFfC	SyoESdzfeJsGhÆPYLTizsKhgovklWFfC	b'gAAAAABkVEgK1CT5Vb6RG-usb9C2-kzeKH3ILDJSBUNr...	Sy

10000 rows × 6 columns

As it is tough to manually check whether all the 10000 inputs are encrypted and decrypted properly, we have compiled these results to the number of "FAILED" cases for each value of "*fuzz_factor*" ranging from 10 to 19 into a table.

For example:

Information Systems Security Project 2

Akhil Pragallapati
Siva Sravana Kashyap Pathiki

```
fuzz_factor_results
```

	Row Number	Fuzz Factor	Number of errors
1	1	10	0
2	2	11	0
3	3	12	0
4	4	13	0
5	5	14	0
6	6	15	0
7	7	16	0
8	8	17	0
9	9	18	0
10	10	19	0

Conclusion:

The Python "**fuzzing**" package's "*fuzz_string*" method offers a quick and effective approach to create random strings for fuzz testing. It is a crucial technique for locating flaws and vulnerabilities in software systems. Developers can make their systems more reliable and secure by employing fuzzing techniques, which will enhance the overall quality of their program.

Finally, based on the above results we can conclude that our encryption and decryption functions are robust and have passed the fuzz testing that has been done using the "**fuzzing**" tool.