Cyber_suite

Motivation/Introduction

I got really interested in hacking and ctfs after the Zense team sent us a mail way back in 1st sem , talking about bandit(https://overthewire.org/wargames/) and ctfs in general. I tried to do them when I got some free time. They were hard ,to be honest, but in the process I noticed that there were similarities and mundane tasks, in challenges that could possibly be automated. So I thought I would build a suite of tools that would help in these hacking challenges like convertion from one form to another, encryption and decryption of basic ciphers and maybe try and detect vulnerabilities in code. Now I know that there are some amazing tools(mostly free!) which do these tasks super efficiently and there is no way that a 1st year cse student who has just started out with hacking and making large projects could match those projects. But I really enjoy these challenges and I thought building tools which might help in them, would be fun. And it was!

The tools I was finally able to accomplish was not as vast I had imagined but I think they are decent: 1. Converter: Converts between hex,decimal,binary and ascii. 2. Encrypter: Given a file or string it encrypts the data using the specified cipher(Ceasar,vignere with specified keys) 3. Decryptor: Given an encrypted file and type of encryption(ceasar,vignere,monoalphabetic random substitution) it decrypts the data to the best possible extent. 4. Static Code Analysis: Given python files it outputs lines which might be vulnerable to sql injection and command line injection.

Modules used

1. sys: For command line arguements
2. ast: Abstract Syntax Tree used for static code analysis

* No special installation of modules required

Usage

Converter

python3 converter.py conversion value_to_be_converted

Eg: `python3 converter.py hex-dec ff`

conversion arguement is of the form datatype1-datatype2(asc,hex,dec,bin)

Eg: hex-asc, dec-bin, asc-bin.... All 12 combination are allowed

Note: Hex values cannot have upper case letters. So A3 is not allowed. Instead use a3.

Encrypter

python3 encrypt.py option cipher key data

Eg: `python3 encrypt.py -s rot 13 sooraj` `python3 encrypt.py -f vignere gold testfiles/test2.txt`

option: -s or -f to input direct string or input file contaning text

cipher: rot(caesar) or vignere

key: For rot it has to be a number and for vignere it has to be a small word.

https://www.geeksforgeeks.org/vigenere-cipher/

https://en.wikipedia.org/wiki/Caesar_cipher

data : if -s then a string has to be given. If -f then name of file has to be given

Decrypter

Broadly speaking there are 2 types of decryption here: where key is known and where key is not known. In the latter case we will try to guess using some user based input and some brute forcing. Examples will hopefully make it clearer.

python3 decrypt.py option cipher data

```
python3 decrypt.py -sk rot 13 fbbenw
```

option: 1. -s : if key is unknown and data is a string given directly 2. -sk: key is known and data is a string given directly. for rot key is a number and vignere it is a word/string 3. -sK: specifically for vignere where only keyword length is known but key itself is unknown. Works best for small keywords and large data. Use -fK instead.

4. Similar where input is a file except change s to f. So -f,-fk,-fK.

cipher: 1. rot: Caesar cipher. If key is unknown then all 26 possible rotations are tried and displayed. Left to user to take the text that makes sense.

```
python3 decrypt.py -s rot fbbenw
```

2. monoalph: Monoalphabetic substitution cipher where each letter is mpped to some other random letter. There is no standard offset like in caesar cipher. For this frequency analysis is used . This needs a bitmore interaction and attention from user using their knowledge of English and using some context.Lets take an example...

In test2.txt there is some text I had written in the start of sem1. It is composed of only English alphabets. test7.txt has this text but in encrypted form.

```
python3 decrypt.py -f monoalph testfiles/test7.txt
```

We first see the original file contents and then *s and then the first iteration of decrypted text, This still seems weird but upon closer inspection we can see the word 'tre' and can make an educated guess that it has to be 'the' . So we then change the mapping of 'g' which initially mapped to 'r' to now map to 'h'.

Alt text

We then see the second iteration of decrypted text. Alt text

We see word 'the' appearing correctly. Now there are more words that are wrong by just 1 letter like 'hsve'->'have' , 'everg':'every', etc. Changing the mapping of 'j' to 'a' instead of 'j' mapping to 's' the text becomes even more understandable.

Keep proceeding in this way to decrypt it fully.

After a few iterations of this we get this:

Alt text

Using some common sense and educated guesses we can undersatnd what is written. "Due to the coronavirus pandemic most institutions have gone virtual....". If full decryption(if something like flag/password is needed) is wanted then few more iterations will reveal the decrypted text

https://overthewire.org/wargames/krypton/krypton3.html

https://axcheron.github.io/writeups/otw/krypton/

Note : It is quite a cumbersome process , I know :(. But frequency analysis is unfortunately not very accurate except for the first few letters like 'e' and 't'. Thats why finding 'the' is probably the best first step. Again large amounts of text are needed for decryption.

vignere: if only keylength is known then it tries to guess the keyword using frequency analysis on letters which are at a distance of 1 keylength.

eg: `python3 decrypt.py -fK vignere 4 test5.txt`

Alt text

Eg: `python3 decrypt.py -fk vignere gold testfiles/test5.txt` , `python3 decrypt.py -sk vignere gold yczugx`

Note: As in encryption non alphabetic characters are ignored and upper case letters are converted to lower case for ease of decryption.

Static Analysis

python3 static_analysis.py file1.py file2.py file3.py.....

```
python3 static_analysis.py testfiles/test.py
```

Goes through all the python files given and displays line numbers which have code that might be sql injection and command line injection vulnerable.

For sqli it just sequentially lists the line numbers .

For command injection it divides the severity ino 3 categories: 1. Critical: Where no input validation/sanitation is done. This is terrible and can be exploited. 2. High 3. Medium: Where some input validation is detected but command injection it is still possible.

Here I have taken the side of caution so number of false positives might be a bit high and 'Medium' and 'High' sometimes don't make sense. It was really hard to accurately verify what was being checked in the if conditions, due to the variability involved in the condition checking process.

https://deepsource.io/blog/introduction-static-code-analysis/

https://rushter.com/blog/detecting-sql-injections-in-python/

https://greentreesnakes.readthedocs.io/en/latest/tofrom.html

https://blog.securityinnovation.com/blog/2011/06/how-to-test-for-command-injection.html

Conclusion

In the end I can say that it realy was a fun project and I got to learn a lot in the process. The Abstract Syntax Tree part was probably the most challenging part. But once I understood it , after reading the doc , I found it extremely powerful. Now I feel I can analyze python code for various things using this ast module. Again I reiterate there are many online and open source tools that do this quite efficiently, but I took up this project up so that I could build a fast ,offline, command line suite of tools that could maybe do the same job.

Future Developments/Scope

1. Enhance Static Analysis to detect other vulnerabilities like xss for javascript,etc.
2. Build tools that can automate Buffer Overflow , Password bruteforcing and so on.(These are outside my skill level right now I think)
3. Add more ciphers for encryption and decryption.

There's always more to do!

test5.txt has encoded text of test2.txt using vignere cipher and key='gold'.