freq.py

Abstract

freq.py is what happens when Mark Baggett sits in on your class, and you dangle interesting problems in front of him…

Background: adversaries attempt to bypass signature based/pattern matching/blacklist techniques by introducing random: filenames, service names, workstation names, domains, hostnames, SSL cert subjects and issuer subjects, etc.

Problem: detecting randomly-generated X is a powerful defensive technique, but hard with a narrow scope.

Solution: sign Mark Baggett up for your class and…freq.py is born

freq.py helps us solve the problem by employing frequency tables that map how likely one character will follow another

Where to Acquire

https://github.com/MarkBaggett/MarkBaggett/tree/master/freq

Examples/Use Case

Note: The higher the number returned by freq.py the more likely it is to occur

---

Blindly freq-ing for EVIL

> Mark provides some pre-built frequency tables built using public domain fiction and speeches as seed text. While not the preferred approach, just using the provided frequency tables can hit pay dirt.

Default Frequency Tables

```
[/opt/freq]$ ls *.freq
english_lowercase.freq  english_mixedcase.freq
```

Using Default Frequency Tables

Measure (`-m`) the likelihood of the characters in the string **`sec511`** occurring in that order:

```
[/opt/freq]$ python freq.py -m "sec511" english_lowercase.freq
5.03581076834
```

Measure (`-m`) the likelihood of the string **`xzkravkdj`**:

```
[/opt/freq]$ python freq.py -m "xzkravkdj" english_lowercase.freq
1.19928269423
```

Bulk (`-b`) measure the likelihood for each entry in **`/home/student/bootcamp/test_domains.txt`**

```
[/opt/freq]$ python freq.py -b /home/student/bootcamp/test_domains.txt english_lowercase.freq
```

Building a Frequency Table

> Rather than using the provided tables, we can instantiate our own. This is the preferred approach, because our fidelity should improve with frequency tables that are built based on normal seed data for our target. For example, if we will be using freq.py to look for random generate executable names, then supplying a large volume of normal executable names would yield better results than just text based on speeches and fiction in the public domain.

Create (**-c**) a new frequency table (in this case 511_domains.freq)

```
[/opt/freq]$ python freq.py -c 511_domains.freq
```

Toggle on (**-t**) case sensitivity (disabled by default)

```
[/opt/freq]$ python freq.py -t 511_domains.freq
Case sensitivity is now set to True
```

Feed (**-f**) the frequency table with representative (read: normal) data

```
[/opt/freq]$ python freq.py -f /home/student/bootcamp/normal_domains.txt 511_domains.freq
```

Tactically freq-ing for EVIL

Measure (**-m**) the likelihood of the string **sec511.com**

```
[/opt/freq]$ python freq.py -m "sec511.com" 511_domains.freq
```

Bulk (**-b**) measure the likelihood for each entry in **/home/student/bootcamp/test_domains.txt**

```
[/opt/freq]$ python freq.py -b /home/student/bootcamp/test_domains.txt 511_domains.freq
```

Tuning Tables

Update frequency table with a normal (**-n**) entry as if seen 10000 times

```
[/opt/freq]$ python freq.py -n "qwerty.sec511.com" -w 10000 511_domains.freq
```

Update frequency table with a file (**-f**) containing normal entries (e.g **/home/student/bootcamp/normal_domains.txt**)

```
[/opt/freq]$ python freq.py -f /home/student/bootcamp/normal_domains.txt 511_domains.freq
```

Update frequency table with an odd (**-o**), bogus, but not random entry

```
[/opt/freq]$ python freq.py -o ".ru" 511_domains.freq
```

---

freq.py command line switches

Additional Info

https://isc.sans.edu/forums/diary/Detecting+Random+Finding+Algorithmically+chosen+DNS+names+DGA/19893/
https://isc.sans.edu/diary/freq.py+super+powers%3F/19903
https://isc.sans.edu/forums/diary/Continuous+Monitoring+for+Random+Strings/20451/