CS5435 - Security and Privacy Concepts in the Wild

Ryan Hall

Homework #3

October 29, 2018

Description of Algorithm 1

In the first algorithm, our goal is to create mutations of the provided passwords in an attempt to confuse a potential adversary. For example, if the provided password was “password123”, we might attempt to create passwords like “password123!”, “password1”, and “Password123”. These need to be sufficiently realistic in order to confuse a potential adversary. Since the honeywords are generated with a degree of pseudo-randomness, some honeywords will be less convincing than others. Therefore, the larger the number of honeywords, the better. We built our algorithm to create the following password mutations:

- Add padding to the end of a password, which could mutate “password” into “password!!!”
- Change the delimiter, which could mutate “i_love_you” into “i-love-you”
- Substitute special characters, which could mutate “password!” into “password#”
- Random tweaks, which could mutate “password” into “paszword”

When generating lists of honeywords, our algorithm randomly selects a mutater, and runs the mutation on the input password. We repeat the process until we have the desired number of honeywords. To add additional entropy we also use a “tough nut” generator, which will occasionally add long, random-looking, honeywords to the list. This may confuse the attacker into thinking that the password was generated from a password manager, or something similar.

Description of Algorithm 2

The second algorithm implements the same password mutations approach, as described above, but additionally utilizes the top 100 passwords from the rockyou database. The passwords from rockyou are used as “seed” passwords, upon which we create several mutations in an attempt to throw off potential adversaries. This approach is useful because an adversary may be able to conclude the most likely password from a subset of mutations (as would be the case using algorithm 1), but if we use multiple seed words, then an adversary must now make 2 decisions: (1) which cluster, or category of honeywords is correct, and (2) within that cluster, which honeyword is the password. We accomplish this by randomly choosing seed words from the rockyou top 100. (We only choose enough such that each seed word has at least 2 mutations in the final list of honeywords.) Then, we perform mutations of those seed words and add them to the list of honeywords. This could result in a list like the following:

```
coffee123, coffee!, coffee, drink_tea!, drink-tea!, drink-tea!#, spinach1, spinach#, zinach
```

From the list above, one can clearly determine the three groups of honeywords, but still only a single honeyword is the actual password.

Description of Algorithm 3

Our 3rd algorithm operates identically to our 2nd algorithm, with one exception: we now use the full rockyou password database. This adds a substantial degree of entropy to our honeyword list, and is significantly more resilient against an adversary. If, hypothetically, we were using algorithm 2 to generate 10 honeywords for 1000 passwords, and that data was leaked, then an adversary could easily use a frequency analysis to discount unlikely honeywords. For example, under the aforementioned conditions, mutations of the password "password123" would occur about 60 times in the leaked data, which is far more likely than any individual password in a population. Using that information, an attacker could rule out those honeywords as being unlikely to be the real password. Now that we are using the full rockyou database in our 3rd algorithm, our honeywords will be much more protected against frequency analysis.