Analiza i ocena bezpieczeństwa systemów usługowych i IoT
Ocena skuteczności różnych metod łamania haseł
Raport 2

Jan Pajdak
Wojciech Słowiński
Maria Filemonowicz

20 maja 2019

Prowadzący: Dr hab. inż. Grzegorz Kołaczek

Spis treści

1 Cel eksperymentu

Hasła tekstowe to obecnie najpopularniejsza metoda uwierzytelniana używana do ograniczania dostępu do zasobów takich jak serwisy internetowe czy konta pocztowe przez osoby nieupoważnione. Zabezpieczenia tego typu są łatwe w użyciu jednakże proste do złamania — w ramach eksperymentu analizowane będzie łamanie haseł przy użyciu algorytmów *BFM* oraz *Weira*.

Eksperyment będzie przeprowadzony przy użyciu bazy realnych haseł, które następnie będą badane pod kątem odporności na złamanie przez poszczególne algorytmy.

2 Plan eksperymentu

2.1 Źródło danych

Jako źródło danych wybrana została baza danych znaleziona w roku 2017 przez firmę z branży cyberbezpieczeństwa - *4iQ* [2]. Baza jest kompilacją informacji z 252 wycieków; zawiera loginy i hasła do ponad 1.4 miliarda kont. Całkowity rozmiar danych to 41.1 GB. Osoba odpowiedzialna za stworzenie bazy danych jest nieznana; dane zostały odkryte przez *4iQ* w *dark web* i można je obecnie pobrać przy użyciu sieci *torrent*.

Dane muszą zostać sformatowane przed użyciem ich w eksperymencie — są one porozdzielane na wiele plików oraz zawierają loginy i adresy poczty elektronicznej powiązane z kontami; te dodatkowe informacje są zbędne. Ze względu na ilość danych badany będzie podzbiór haseł.

2.2 Planowany przebieg eksperymentu

Algorytmy rozważane w niniejszym eksperymencie posiadają wysoką złożoność obliczeniową i czasową, przez co testowanie ich implementacji na dużych zbiorach haseł byłoby bardzo ograniczone przez moc obliczeniową dzisiejszych komputerów i czas potrzebny na przeprowadzenie takich badań. Aby umożliwić przeprowadzenie eksperymentu na o wiele większej bazie haseł, wykorzystane zostaną metody pozwalające na wyliczenie liczby prób potrzebnych do odgadnięcia hasła w obu algorytmach, nazywane *kalkulatorami* [3], dzięki którym wyznaczyć można siłę hasła.

2.3 Technologie

Do formatowania bazy haseł wykorzystany został *Python*.

Algorytmy oceniające skuteczność łamania haseł w poszczególnych algorytmach zostały zaimplementowane przy użyciu *Scala*.

2.4 Metoda oceny

2.4.1 BFM

Kalkulator oceny skuteczności algorytmu *BFM* działa następująco:

1. Na podstawie treningowego zbioru haseł określane są:

   Zbiór pojedynczych znaków alfabetu wraz z częstotliwością ich występowania jako pierwsza litera hasła

   Zbiór wszystkich możliwych digramów ułożonych ze znaków w alfabecie wraz z częstotliwością ich występowania, tworzony w następujący sposób:

   Zakładając zbiór treningowy złożony ze znaków *A, B, C*; Jeżeli znak A ma największe prawdopodobieństwo wystąpienia jako pierwszy, znak *B* ma największe prawdopodobieństwo wystąpienia po *A* a znak *C* ma największe prawdopodobieństwo wystąpienia po *B*, pierwszą próbą odgadnięcia hasła będzie *ABC*.

2. Na podstawie powstałych zbiorów określona zostaje kolejność zgadywania kolejnych znaków w haśle

3. Dla właściwego zbioru haseł wyznaczana jest liczba wymaganych prób potrzebnych do jego odgadnięcia:

   N - długość zbioru znaków w alfabecie

   L - długość zgadywanego hasła

   M - minimalna długość hasła

   CF - liczbę znaków sprawdzanych przed odgadnięciem właściwego pierwszego znaku hasła

   DF(i) - liczba digramów sprawdzanych przed odgadnięciem właściwego i-tego znaku hasła

   i - pozycja znaku w haśle

   k - k-ta próba odgadnięcia hasła

   X - liczba haseł krótszych niż zgadywane

   Y - liczba haseł z niepoprawnie odgadniętą pierwszą literą

   Z - liczba haseł z niepoprawnie zgadniętym i-tym znakiem

   Q - liczba prób potrzebnych do odgadnięcia zadanego hasła

   $X = \sum_{l=L}^{l=M} N^l$

   $Y = CF * N^{L-1}$

   $Z = \sum_{i=2}^{i=L} DF(i) * N^{L-(i+2)}$

   $Q = X + Y + Z$

   Jeżeli pierwszy znak nie zostanie odgadnięty poprawnie to wiemy, że algorytm podejmie N^{L-1} prób, zanim spróbuje odgadnąć hasło z innym znakiem na pierwszej pozycji

   Zgodnie z powyższym, dla k-tej próby odgadnięcia pierwszego znaku wiemy, że algorytm podejmie co najmniej $(k-1)N^{L-1}$ prób odgadnięcia hasła

2.4.2 Weir

Pojęcia używane w algorytmie Weir'a:

- struktura (*structure*) - opis określający typ każdego znaku w haśle

  L - litera

  D - cyfra

  S - znak specjalny

- (*terminal*) - instancja (hasło) spełniająca założenia struktury

- grupa prawdopodobieństwa (*probability group*) - grupa terminali o jednakowym prawdopodobieństwie wystąpienia w zbiorze treningowym

Przykład: struktura hasła "password123!@#" to "LLLLLLLLDDDSSS"

Proces tworzenia tablicy z której korzysta algorytm jest bardziej złożony niż w przypadku algorytmu BFM.

Algorytm 1: Pseudokod opisujący proces tworzenia tablicy używanej w kalkulatorze Weir'a

$T \leftarrow$ nowa tablica
forall *strukturys* **do**
  forall *grupy_prawdopodobiestwapg* **do**
    forall $lcs \in pg$ **do**
      c_i = liczba wystąpień lcs
      p_i = prawdopodobieństwo wystąpienia lcs w zbiorze treningowym
    end forall
    $probability = \prod p_i$
    $size = \prod c_i$
    $T.add: \quad pg, probability, size$
  end forall
end forall
$Sort(T)$ by $probability$

2.4.3 Estymator siły haseł Monte Carlo

W celu porównania ze sobą różnych algorytmów, służących do łamania haseł, skorzystano również z estymatora Monte Carlo [4], który szacuje liczbę prób potrzebnych do zgadnięcia danego hasła. Na podstawie ciągu treningowego wybierana jest próbka o zadanej długości n haseł na podstawie prawdopodobieństwa ich zgadnięcia w każdym z modeli ataków, wśród której estymator oszacowuje ile haseł z próbki zostanie złamane przed złamaniem podanego hasła, ustalając tym samym przybliżoną liczbę prób potrzebną na jego złamanie. Metoda ta nie jest dokładna, a wyniki są przybliżone i często zależą od różnych parametrów (takich jak wielkość próbki, wielkość ciągu treningowego), jednak pozwala na porównanie wielu modeli ataków oraz różnych polityk tworzenia haseł.

W implementacji Monte Carlo [1] wyróżnione zostały modele ataków bazujące na n-gramach, będące rozszerzeniem algorytmu BFM o dłuższe ciągi znaków, oraz model PCFG, który zaproponowany został przez Weira.

3 Przebieg eksperymentu

3.1 Przygotowanie danych

Dane znajdujące się w pobranej bazie są podzielone na 1 981 plików, w 54 folderach. Informacje w plikach są przechowywane w formacie *nazwa_użytkownika:hasło*. Rozmiar plików nie jest stały. Przed przystąpieniem do eksperymentu za pomocą skryptu napisanego w języku *Python* dane zostały uporządkowane tak, by utworzyć łatwy do analizy zbiór haseł. Algorytm wykonuje następujące operacje dla każdego pliku wchodzącego w skład pobranej bazy:

1. Wczytuje pojedynczą linię z pliku
2. Usuwa część linii zawierającą hasło
3. Przechodzi do następnej linii jeżeli hasło zawiera znaki inne niż:

   cyfry

   litery z alfabetu łacińskiego

   znaki specjalne

Tak przygotowana linia jest następnie przydzielana do kategorii i zapisana do odpowiedniego pliku. W eksperymencie rozróżniamy trzy kategorie haseł:

1. *raw* - każde hasło które nie zostało odrzucone po analizie znaków
2. *regular8* - hasło o długości min. 8 znaków
3. *comprehensive8* - hasło o długości min. 8 znaków oraz zawierające co najmniej 1 dużą literę, 1 cyfrę i 1 znak specjalny

Wymogi kategorii *regular8* oraz *comprehensive8* bazują na popularnych wymogach dotyczących haseł stosowanych w różnych serwisach internetowych.

Zaledwie *0,5%* haseł wchodzących w skład bazy spełnia wymogi *comprehensive8*.

3.1.1 Ciąg treningowy i ciąg walidacyjny

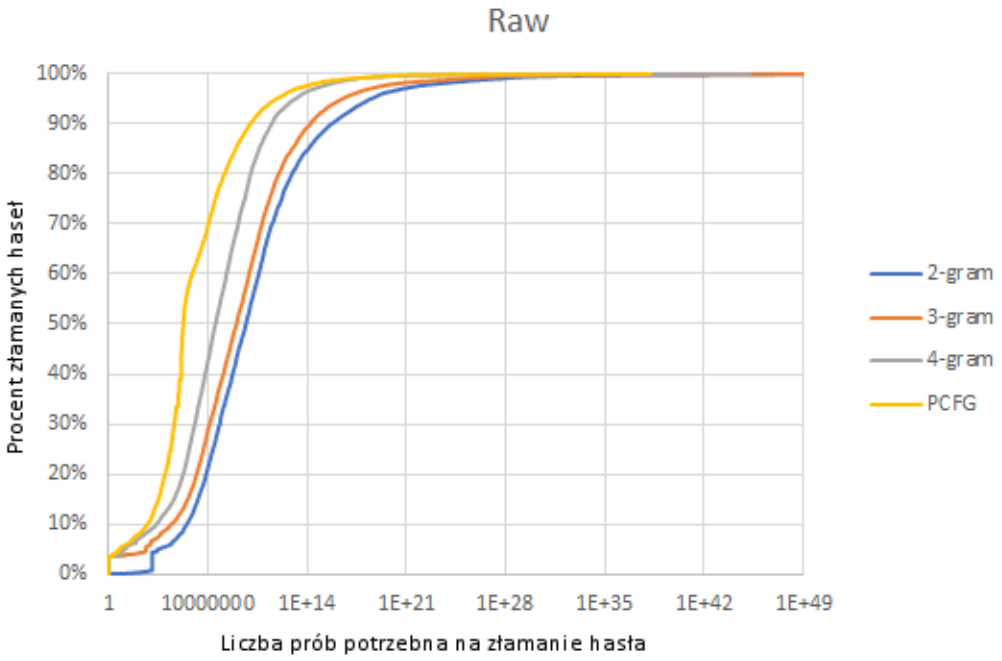
Ciąg treningowy to ważna część danych wejściowych, od których zależy skuteczność algorytmów. Dla każdej kategorii haseł został utworzony zbiór składający się z losowo wybranych haseł z odpowiadającej kategorii. Podobnie zostały stworzone ciągi walidacyjne.

Ciąg treningowy dla każdej kategorii haseł składa się z 500 tysięcy haseł, natomiast ciąg walidacyjny z 50 tysięcy haseł.

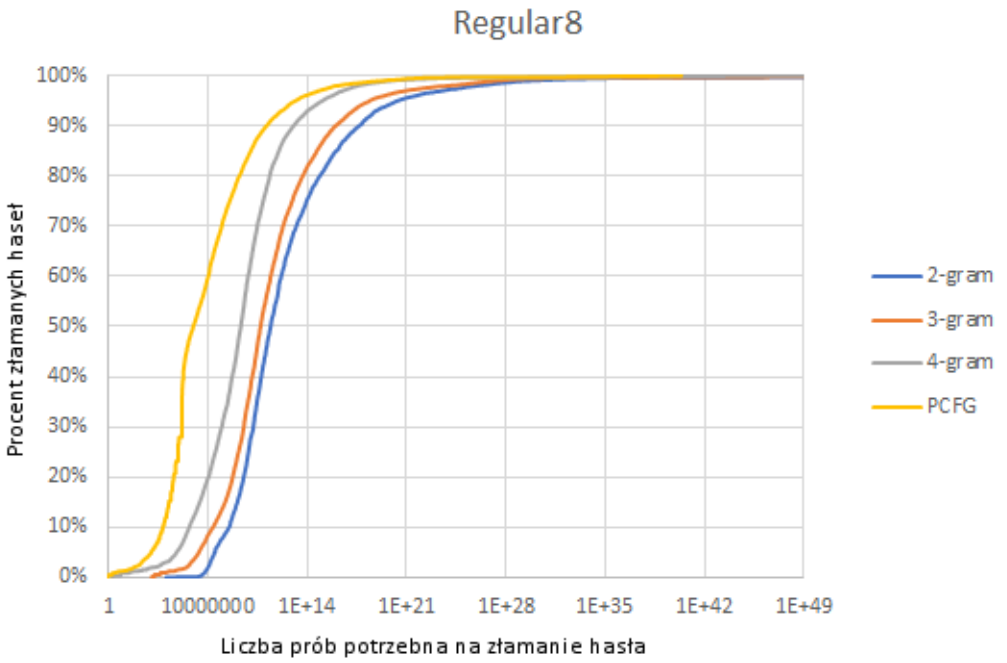
Dane służące jako zbiór treningowy kalkulatora oceny skuteczności algorytmu Weir'a zostały dodatkowo uporządkowane przy użyciu skryptu. Hasła zostały rozdzielone do plików odpowiadających ich strukturom. Dodatkowo, utworzony został plik zliczający wystąpienia każdego *lcs*.

4 Wyniki

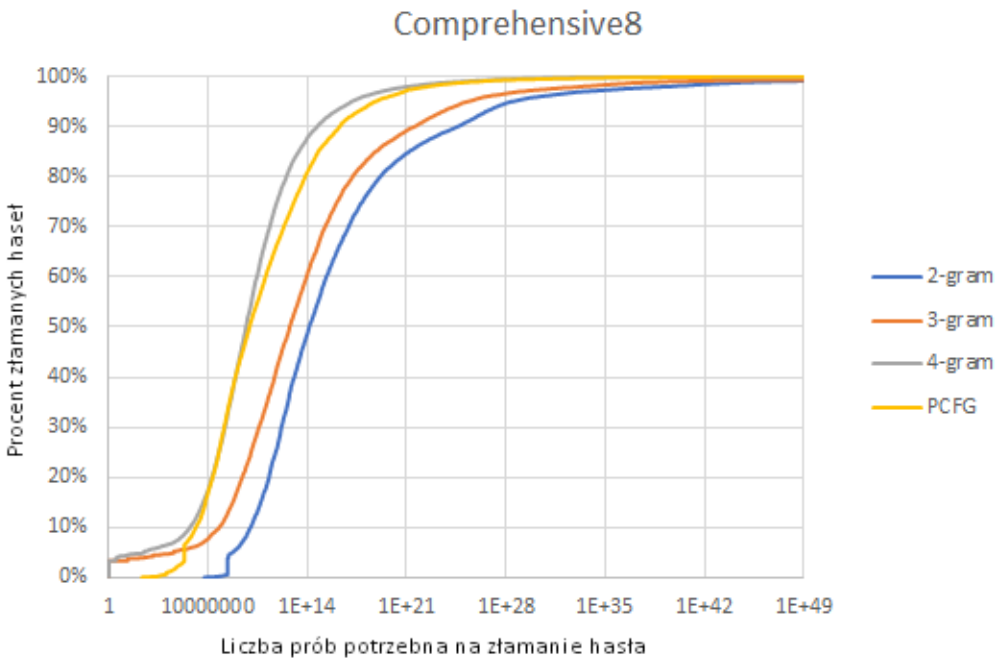
4.1 Porównanie wpływu polityki tworzenia haseł na ich siłę w kontekście różnych algorytmów



Rysunek 1: Wpływ rodzaju algorytmu na liczbę prób potrzebnych do złamania haseł nienależących do żadnej polityki



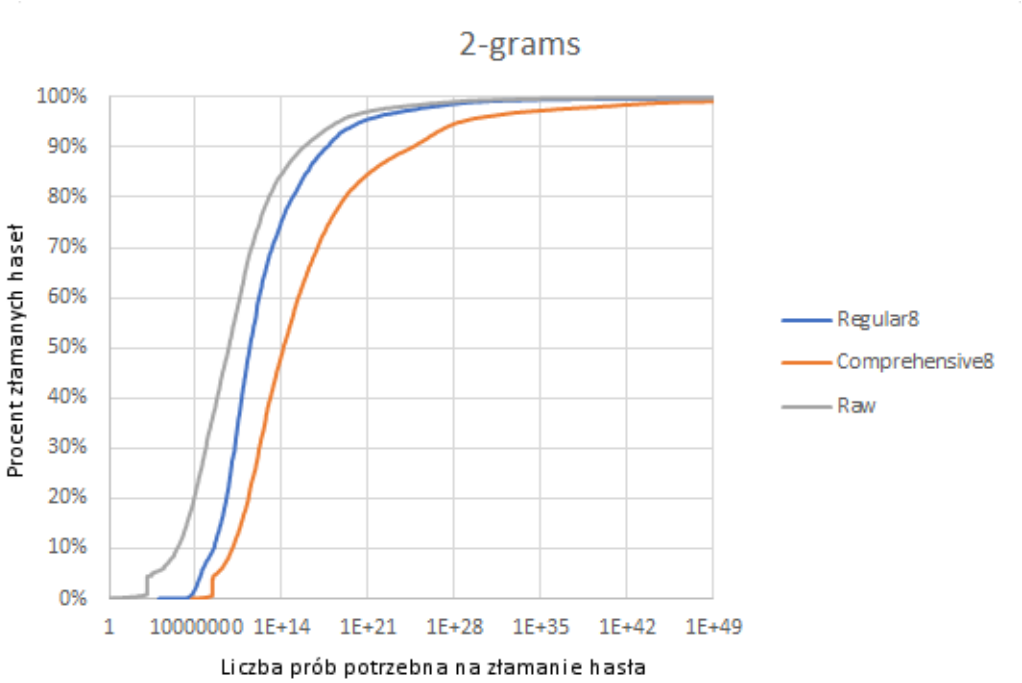
Rysunek 2: Wpływ rodzaju algorytmu na liczbę prób potrzebnych do złamania haseł w polityce regular8



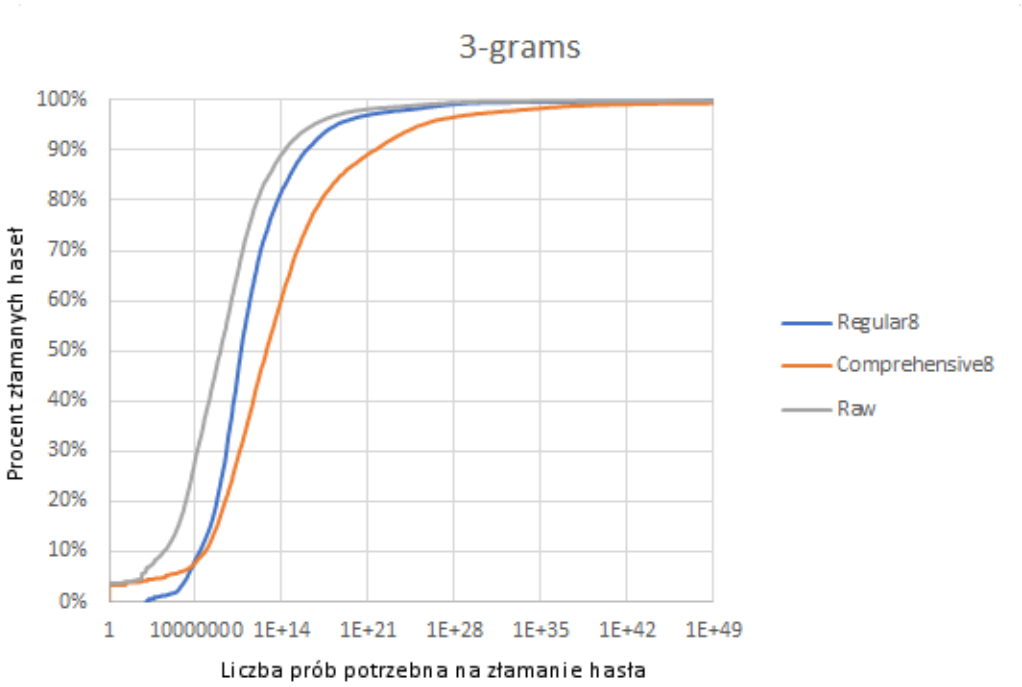
Rysunek 3: Wpływ rodzaju algorytmu na liczbę prób potrzebnych do złamania haseł w polityce comprehensive8

Rysunki 1, 2 i 3 przedstawiają ile faktycznych prób musi wykonać każdy z algorytmów, aby odgadnąć procentową część wszystkich haseł z ciągu walidacyjnego.

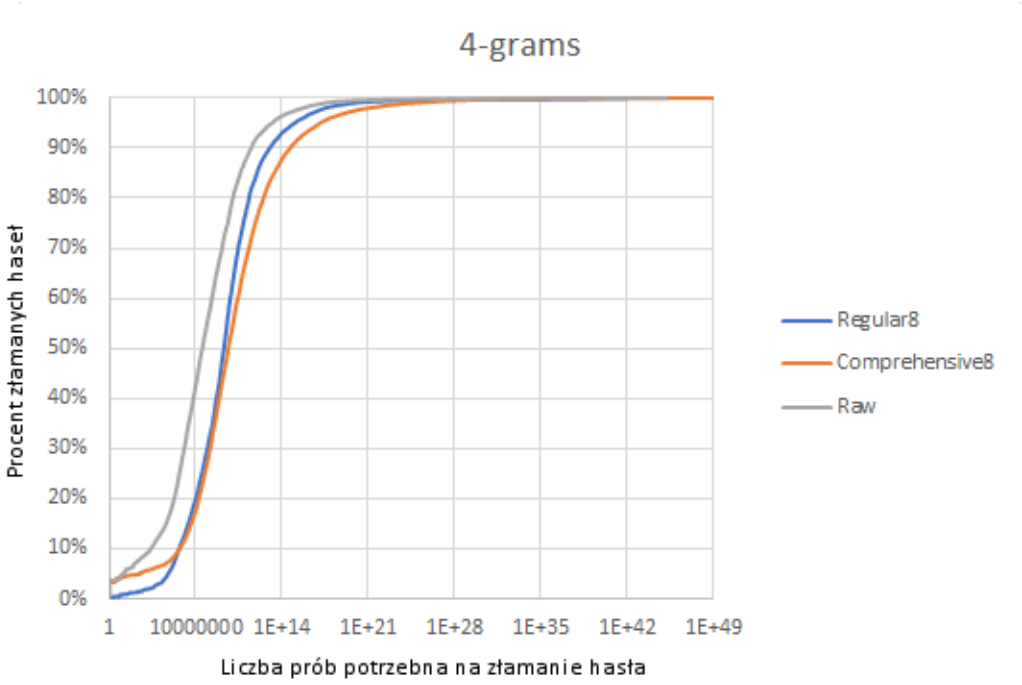
4.2 Porównanie wpływu rodzaju algorytmu na skuteczność łamania haseł w każdej z polityk



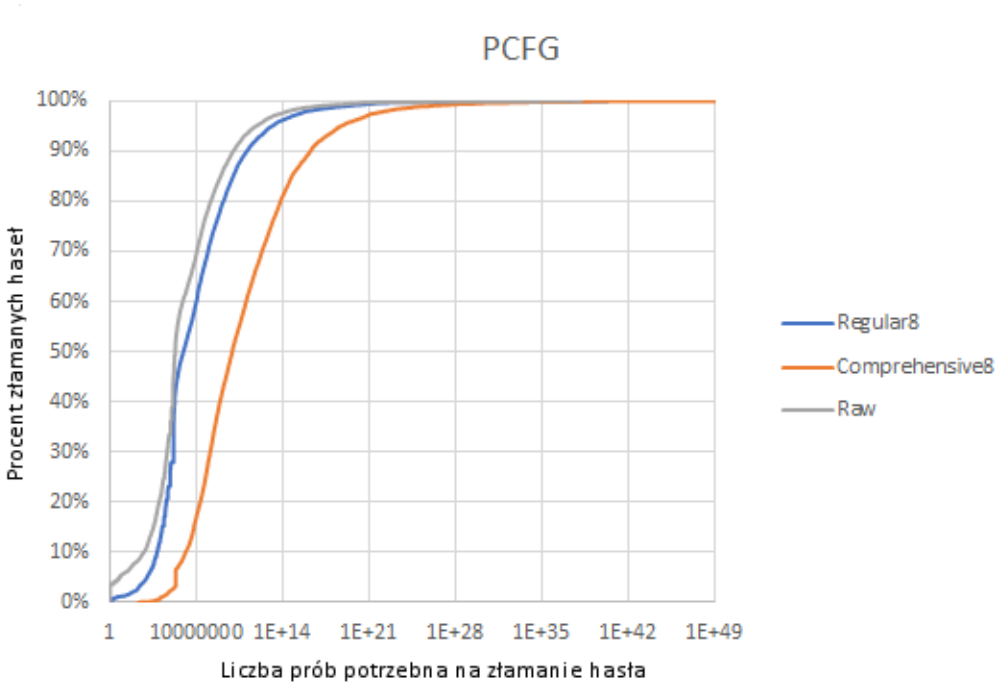
Rysunek 4: Skuteczność łamania haseł dla każdej polityki w algorytmie bazującym na 2-gramach (standardowy BFM)



Rysunek 5: Skuteczność łamania haseł dla każdej polityki w algorytmie bazującym na 3-gramach



Rysunek 6: Skuteczność łamania haseł dla każdej polityki w algorytmie bazującym na 4-gramach



Rysunek 7: Skuteczność łamania haseł dla każdej polityki w algorytmie PCFG (algorytmie Weira)

Rysunki 4, 5, 6 i 7 przedstawiają liczbę prób potrzebną do złamania procentowej części wszystkich haseł z ciągu walidacyjnego.

5 Analiza wyników

5.1 Analiza wpływu polityki tworzenia haseł i rodzaju używanego algorytmu na skuteczność łamania haseł

Przeprowadzone badania potwierdzają zależność pomiędzy stopniem złożoności algorytmu a jego skutecznością niezależnie od wybranej polityki tworzenia haseł. Algorytm PCFG, który przeprowadza najbardziej szczegółową analizę haseł z ciągu treningowego, wyprzedza wszystkie algorytmy bazujące na n-gramach w liczbie prób, potrzebnych do złamania hasła. Podczas porównywania ze sobą n-gramów, również obserwowany jest wpływ długości ciągów znaków na skuteczność algorytmu - niezależnie od rodzaju łamanych haseł dłuższy ciąg znaków skutkował większą efektywnością.

W oparciu o rysunek 3 przy polityce comprehensive8 można zauważyć przewagę w skuteczności algorytmu opierającego się na 4-gramach nad algorytmem PCFG w ponad *50%* badanych haseł z ciągu walidacyjnego. Aby potwierdzić zależność pomiędzy tą konkretną polityką a skutecznością obu algorytmów, warto wykonać powtórne testy na większym zbiorze treningowym.

Zarówno w przypadku algorytmów opierających się o 2-gramy, jak i 3-gramy, widoczna jest różnica wpływu polityki tworzenia haseł na ich siłę (w kontekście ich łamania przez w. w. algorytmy). Hasła, które nie miały żadnych zasad regulujących strukturę ich tworzenia, były bardziej podatne na złamanie niż w przypadku polityki regular8 czy comprehensive8. Wśród tych dwóch polityk z kolei w około *90%* przypadków silniejsze były hasła, których struktura zawierała minimum jeden znak specjalny, jedną cyfrę i jedną wielką literę.

W przypadku algortmów opierających się na 4-gramach różnice w sile haseł w zależności od polityki są o wiele mniejsze niż w przypadku 2-gramów czy 3-gramów. Wciąż można zaobserwować taką samą tendencję, jednak w szczególności polityki regular8 i comprehensive8 zbliżają się do siebie swoją podatnością na złamanie.

W przypadku algorytmu PCFG różnice w sile haseł są nieco większe - widoczna jest duża przewaga haseł w polityce comprehensive8 nad innymi politykami. Zacierają się jednak różnice w podatności na łamanie haseł z kategorii raw i regular8.

6 Podsumowanie

Literatura

[1] *Monte Carlo password checking implementation*. URL `https://github.com/matteodellamico/montecarlopwd`. Dostęp 10.05.2018.

[2] Julio Casal. *1.4 Billion Clear Text Credentials Discovered in a Single Database*, 2017. URL `medium.com/4iqdelvedeep/1-4-billion-clear-text-credentials-discovered-in-a-single-database-3131d0` Dostęp 08.04.2018.

[3] Patrick Gage Kelley Saranga Komanduri Michelle L. Mazurek Richard Shay Timothy Vidas Lujo Bauer Nicolas Christin Lorrie Faith Cranor and Julio Lopez. *Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms*, 2012. URL `archive.ece.cmu.edu/ lbauer/papers/2012/oakland2012-guessing.pdf`. Dostęp 08.04.2018.

[4] Maurizio Filippone Matteo Dell'Amico. *Monte Carlo Strength Evaluation: Fast and Reliable Password Checking*, 2015. URL `http://www.eurecom.fr/ filippon/Publications/ccs15.pdf`. Dostęp 10.05.2018.