BDAN

Projekt: Profesjonalny algorytm kryptograficzny

Dominika Atroszczyk, Daria Shevchenko

MAKWA:

bezpieczna funkcja haszująca hasła

Język: PYTHON

–Czym jest Makwa?

- zaawansowana funkcja haszująca
- stworzona do przekształcenia haseł w tokeny do weryfikacji haseł
- kluczową cechą jest wsparcie dla delegacji

–Czym jest delegacja

Delegacja polega na przenoszeniu kosztownych obliczeń kryptograficznych z klienta na serwer

- **Cel:** utrudnienie ataków, zachowanie niskich kosztów obrony

—Jak działa sól?

Sól to losowy element danych dodawany do hasła przed jego haszowaniem

- **Unikalność:** dla każdego hasła generowana jest unikalna sól
- **Losowość:** nawet identyczne hasła będą miały różne wartości hash
- **Bezpieczeństwo:** chroni przed atakami słownikowymi i atakami na wiele kont

  - **Przykład:**
    - Hasło: **haslo123**
    - Sól: **aB1cD2**
    - Zhashowane hasło: **Hash(haslo123aB1cD2)**

– Kwadratowanie modulo

- Operacja matematyczna, w której liczba jest podnoszona do kwadratu, a następnie wynik tej operacji jest dzielony przez inną liczbę (modulus). Wynikiem jest reszta z tej operacji.

$$y = x^2 \mod n$$

— HMAC

- Hash-based Message Authentication Code (HMAC):
  Metoda uwierzytelniania wiadomości przy użyciu kryptograficznych funkcji haszujących w połączeniu z tajnym kluczem

  Umożliwia nam sprawdzenie, czy wiadomość pochodzi od autentycznego nadawcy i nie została zmodyfikowana podczas transmisji.

- Przykład:
  Klucz: “secret_key”
  Wiadomość: “message”

  Hashowanie: HMAC: “SHA-256(secret_key XOR message)”

— KDF

- Funkcja KDF (key Derivation Function) generuje klucz z hasła i soli
- Zapewnia, że nawet jeśli dwa hasła są identyczne, ale mają różne wartości soli, wynikowe klucze będą zupełnie inne.
- Używa funkcji HMAC do przetwarzania danych wejściowych.

Parametry: dane wejściowe, długość wyjściowa klucza, HMAC(key, message, f.hash).

– KDF

- algorytm:

  R - długość skrótu zwracana przez określoną f.hash (np. sha-256)

  V = “1” * R (np. 1111 dla R = 4)

  K = “0” * R (np. 0000 dla R = 4)

  K = HMAC na (R, V+0+data, f.hash)

  np.(1111+0+”data” = 11110data)

  V = HMAC na (K, V, f.hash)

  K = HMAC(K, V+1+data, f.hash)

  ...

PĘTLA

V = HMAC(K, V, digestmod)

T= T+V

While len(T) < out_len

– Parametry algorytmu

Wprowadzane przy zabezpieczaniu

- Password (np. “haslo”)
- Salt (np. “sol”)
- Koszt: m_cost (np. 1000):liczba iteracji kwadratowania)
- Pre-hashing (tak/nie): opcjonalne haszowanie wstępne
- Długość hasha po procesie (np. 100)

Po stronie serwera

- Funkcja hash (np. sha-256)
- Modulus (blum integer)

Modulus = $p \cdot q$, gdzie p i q - liczby pierwsze w formacie $4t+3$, $t \in Z$

Np. p = 3 (t = 0), q =11 (t = 2). Modulus = 3 ▪ 11 = 33.

– Proces algorytmu

- Weryfikacja Modulus:
  Sprawdzamy, czy długość modulus w bitach ≥ 160 bitów. Jeśli mniej, algorytm nie jest wykonywany
- Pre-hashing:
  Opcjonalnie wykonujemy KDF na haśle, jeśli ***pre_hashing = True***
- Sprawdzenie długości hasła:
  255 < dł.hasła w bajtach
  dł. Modulus w bajtach - 32 < dł. Hasła w bajtach

— Proces algorytmu

- Generowanie SB:
  SB = KDF na stringu: salt+password + (dł.password)
  Przykład: “sol” + “haslo” + 5 -> SB = “solhaslo5”
- Generowanie XB:
  XB = 0 + SB + password + (dł.password)
  Przykład: 0 + “solhaslo5” + “haslo” + 5 -> XB = “0solhaslo5haslo5”
- Generowanie X:
  X = bigint(XB), czyli zamieniamy XB na wartość liczbową

W oryginalnym kodzie nie ma żadnych liter, wszystko zapisane jest na bajtach, “solhaslo5” to tylko przykład dla czytelności.

– Proces algorytmu

Proces obliczeń kwadratowania modulo

- W przypadku m_cost = 2, proces wygląda następująco:

Pierwsze kwadratowanie:

$$x_1 = X B^2 \mod \text{modulus}$$

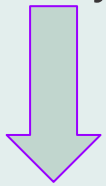
Drugie kwadratowanie:

$$x_2 = x_1^2 \mod \text{modulus}$$

x_2 jest skracane do tej samej liczby bajtów, na ilu jest zapisany modulus:
x_2 = /x55/x66/x77, modulus = /x22/x39 -> x_2 = /x55/x66

— Proces algorytmu

Na x_n wykonujemy KDF, który ostatecznie skraca x_n do długości podanej w parametrach:
post-hashing-length = 100 -> skraca x_n do 100B
post-hashing-length = 0 -> nie robimy KDF i zostawiamy bez skracania



KONIEC

— Implementacja

main.py

Główny skrypt uruchamiający algorytm Makwa. Obsługuje komunikację z “serwerem” w kontekście delegacji obliczeń

makwa.py

Zawiera implementacje algorytmu Makwa. Wykonuje operacje haszowania, kwadratowania modulo, oraz zarządza całym procesem obliczeń.

utilities.py

Zawiera różne funkcje pomocnicze, które wspomagają działanie głównego algorytmu Makwa.

Given inputs:

- password: bdan projekt końcowy :)
- salt: makwa w pythonie
- number of mod squarings: 1000
- pre-hashing enabled: True
- post-hashing length: 100

Server-wide parameters:

- blum integer(n): 351022999992329959805948058429179137071493498564005201690270847802645677768345790194967292172234920966738072206473934607302530155231149611366822488324779862550359247705945490073415356040463966056192182734972882735193738141463240078482438030065572860524108233684186558616800182205050716504974960684274903694268739329364426113890015149870585388182704155583596315748606210332410451928804706233707927619083497201874813615511781886907432805612809508823268899891209780624476214502411841120379122441399868990031856936939031108078845928001314024359275259573951215714002984243912232599737765387970108271965656205

- hash function: sha256

Outputs:

- password in hex bytes: 6264616e2070726f6a656b74206b6fc584636f7779203a29
- salt in hex bytes: 6d616b7761207720707974686f6e6965
- byte length of modulus(k): 251
- pre-hashed password: cf71afad8338f9e34df80865ab41fa58
- byte length of password(u): 32
- SB: 25dab8a09279f7f368491322f0190fc630a1bbc3106189e23f1b95a4215db9615823828a4323bcb60db2756b39be73f54b55... (+217)
- XB: 025dab8a09279f7f368491322f0190fc630a1bbc3106189e23f1b95a4215db9615823828a4323bcb60db2756b39be73f54b5... (+252)
- XB as integer(x): 256461628610927966766368491322660190666363061162626331061896523661629561421564629615823828614323626 3... (+345)
- after modulo-squaring: 244329143550783807931065950915476484542395499811406450493220669397663376045067776096713789975516990 6... (+603)
- post-hashed password: e9459928572fcda63c9cdaa46eab77bff3bdfa46ad1a1e62af2cd3787a9013873107b13f190c88e137c44eade0b38023028c

♥ final result: e9459928572fcda63c9cdaa46eab77bff3bdfa46ad1a1e62af2cd3787a9013873107b13f190c88e137c44eade0b38023028c

---------- MAKWA password hashing function ----------

MAKWA:

bezpieczna funkcja haszująca hasła

Dominika Atroszczyk, Daria Shevchenko