Salted Hash Cracking dengan *Brute Force*

Memecahkan salted hash SHA1 dengan strategi brute force

M Algah Fattah Illahi 13517122
Teknik Informatika
Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung
Bandung, Indonesia
13517122@std.stei.itb.ac.id

***Abstrak*—Pada era teknologi seperti sekarang, ketika jutaan paket berisi data penting beredar setiap menit, keamanan jaringan menjadi sesuatu yang sangat penting. Dalam mengamankan data dari tangan-tangan yang tidak diinginkan, *hashing* menjadi salah satu jawabannya. Sekarang ada begitu banyak macam algoritma *hashing,* salah satunya adalah algoritma SHA-1 (Secure Hash Algorithm 1). Pada makalah ini penulis akan membahas penerapan strategi *brute force* dalam memecahkan SHA1 dari sebuah soal *capture the flag*.**

Kata kunci—hash, salt, brute force, sha1

I. Pendahuluan

Dewasa ini, perkembangan teknologi semakin pesat. Kita dapat mengirim pesan ke belahan dunia lain dalam beberapa milidetik saja. Bayangkan pada zaman dahulu orang harus menunggu berhari-hari bahkan bertahun-tahun untuk mendapatkan kabar dari orang terdekatnya, biaya pengirimannya pun tidak murah. Sebuah nikmat yang sepatutnya kita syukuri untuk dapat menikmati kemudahan ini.

Dibalik kemudahan tersebut, ternyata terdapat kecemasan yang menghantui pengirim dan penerima pesan, yaitu rasa cemas akan kerahasiaan pesan yang mereka kirim atau terima. Begitu banyak informasi yang lalu-lalang di internet, mulai dari informasi yang bersifat sepele yang disampaikan dua orang teman hingga informasi yang mengandung rahasia suatu negara atau organisasi.

Internet bisa diakses oleh siapa saja, sehingga pada dasarnya bisa saja seseorang melancarkan serangan *man in the middle* dan mencuri informasi penting milik anda. Ketika ini terjadi, kriptografi adalah pertahanan terakhir kita.

Saat ini ada berbagai macam kriptografi, mulai dari kriptografi sederhana hingga kriptografi dengan kompleksitas yang sangat tinggi. Kemudian ada kriptografi simetrik yang menggunakan kunci yang sama dalam enkripsi dan dekripsinya, dan ada kriptografi asimetrik yang menggunakan kunci *public* untuk enkripsi dan kunci *private* untuk dekripsi. Dikenal juga istilah *one-way function* dan *two-way function* tergantung apakah *cipher text* bisa dikembalikan menjadi *plain text* kembali atau tidak.

Pada makalah ini, penulis akan membahas sebuah algoritma kriptografi, SHA1 yang merupakan sebuah algoritma *hashing,* dan cara memecahkannya dengan menggunakan pendekatan *brute force*.

II. Teori Dasar

A. *Brute force*

Brute force merupakan pendekatan yang *straightforward* dalam pemecahan masalah. Strategi ini memecahkan persoalan dengan sangat sederhana, langsung, dan jelas. Meski strategi ini terbukti optimal untuk hampir setiap persoalan, algoritma ini memiliki masalah terkait efisiensi karena ia membutuhkan jumlah komputasi yang besar dan waktu yang lama dalam penyelesaiannya.

Beberapa persoalan bahkan hanya bisa diselesaikan dengan metode ini, termasuk *hash cracking* yang akan dibahas dalam makalah ini.

Karena dijamin memberikan hasil yang optimal untuk hampir seluruh persoalan, *brute force* juga sering digunakan sebagai pembanding *algoritma* lain. Dengan membandingkan hasil yang diberikan oleh algoritma *brute force* dengan algoritma lain, kebenaran dari algoritma tersebut dapat diuji dan begitu pula dengan kompleksitasnya. Algoritma yang dibandingkan tentu diharapkan memiliki kompleksitas yang lebih baik dari *brute force* baik dari segi waktu maupun memori.

B. *Fungsi hash*

Fungsi *hash* adalah sebuah fungsi yang menerima masukan *string* dengan panjang sembarang dan mentransformasikannya menjadi *string* keluaran dengan panjang tetap.

Hash merupakan fungsi kriptografi *one-way,* yang berarti *string* yang sudah ditransformasi dengan fungsi ini menjadi *message digest* tidak dapat dikembalikan menjadi pesan semula. Dengan mentransformasi *string* lewat fungsi *hash,* kerahasiaan *string* dapat dijaga.

Fungsi *hash* memiliki banyak peranan dalam kehidupan sehari-hari, diantaranya adalah menjaga integritas data karena fungsi hash sangat peka terhadap perubahan 1 bit pada pesan, menghemat waktu pengiriman karena *message digest* relatif jauh lebih pendek dari pesan asli, dan menormalkan panjang data yang beraneka ragam karena *string* hasil transformasi memiliki panjang yang sama.

Collision adalah kondisi dimana dua *string* sembarang yang berbeda memiliki nilai *hash* yang sama. Hal ini menandakan bahwa fungsi *hash* tidak aman secara kriptografis.

Jika *collision* terjadi, maka *string* berbeda yang menghasilkan *message digest* yang sama akan dianggap sama, sehingga penyerang tidak perlu tahu *string* asli yang diincarnya

Pengamanan data dengan *hashing* biasanya digunakan ketika komunikasi yang terjadi hanya satu arah, sehingga tidak perlu dilakukan dekripsi terhadap *message digest* yang pada dasarnya memang tidak mungkin.

C. Salt

Salt adalah data acak yang disisipkan pada data sebelum ditransformasi lewat fungsi hash. Penambahan *salt* pada *string* dapat dilakukan dengan berbagai cara seperti, ditambahkan di awal sebelum *plain text*, ditambahkan di akhir setelah *plain text* atau bahkan disisipkan di tengah-tengah *plain text.*

Dengan menambahkan *salt* pada *plain text*, penyerang akan semakin kesulitan untuk mendapatkan *string* asli dari *message digest* hasil transformasi fungsi *hash.*

D. SHA1 (Secure Hash Algorithm)

SHA1 merupakan sebuah fungsi kriptografi *hash* yang menerima suatu *input* dan menghasilkan *output* berupa 160-bit (20-*byte*) nilai *hash* yang dikenal sebagai *message digest* yang biasanya diterjemahkan sebagai bilangan heksadesimal dengan panjang 40 digit. Algoritma ini dirancang oleh NSA (National Security Agency).

Panjang pesan maksimum yang dapat diterima algoritma ini adalah adalah 2^{64} bit (2.147.483.648 *gigabyte*).

SHA membutuhkan 5 buah penyangga (*buffer*) dengan panjang masing-masing penyangga adalah 32 bit. Total panjang penyangga yang digunakan adalah 5 x 32 bit = 160 bit.

Kelima penyangga ini diberi nam A, B, C, D, E. Kelimanya diinisialisasi dengan nilai A = 67452301, B = EFCDAB89, C = 98BADCFE, D = 10325476, dan E = C3D2E1F0.

Proses H_{SHA} terdiri dari 80 buah putaran, dimana masing-masing putaran menggunakan bilangan penambah K_t.

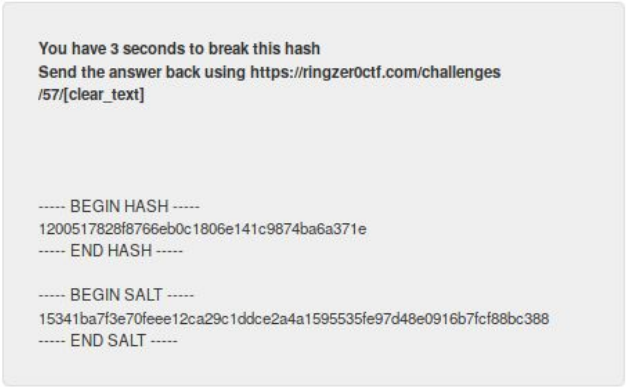
Nilai dari masing-masing bilangan penambah adalah, 5A827999 untuk putaran 0 sampai 19, 6ED9EBA1 untuk putaran 20 sampai 39, 8F1BBCDC untuk putaran 40 sampai 59, dan CA62C1D6 untuk putaran 60 sampai putaran ke 79.

SHA1 dianggap sebagai penerus dari algoritma MD5, dimana MD5 hanya menggunakan 4 putaran sedangkan SHA1 menggunakan jauh lebih banyak putaran, yaitu 80.

Pada tahun 2005, Rijmen dan Oswald mempublikasikan serangan pada versi SHA-1 yang direduksi (hanya menggunakan 53 putaran dari 80 putaran) dan menemukan kolisi dengan kompleksitas sekitar 2^{80} operasi. Pada Februari 2005, Xiayoun Wang, Yiqun Lisa Yin dan Hongbo Yo mempublikasikan serangan yang dapat menemukan kolisi pada versi penuh SHA-1, yang membutuhkan sekitar 2^{69} operasi.

III. ANALISIS

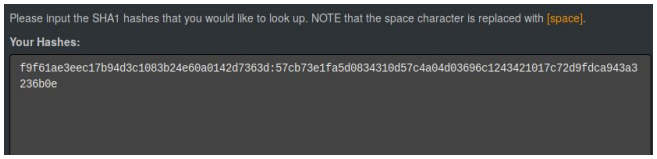
Penerapan strategi *brute force* yang akan dibahas pada makalah ini adalah *cracking SHA1 hash,* untuk menyelesaikan sebuah soal *capture the flag* yang diberikan oleh ringzer0team.com. Soal ini diberi nama "Hash breaker reloaded", dimana peserta diminta untuk memberikan jawaban berupa *plain text* dari *hash* yang sudah diberi *salt* yang diberikan dalam waktu 3 detik.



Gambar 1. *Screenshot* soal

Sumber : https://ringzer0team.com/challenges/57

Untuk menyelesaikannya, pertama kita harus tahu algoritma hash yang digunakan. *Hash* yang diberikan memiliki panjang 40 karakter, yang berarti fungsi *hash* ini mungkin adalah SHA-0, SHA-1, SHA-512, dan WHIRLPOOL. Untuk memastikannya kita akan menggunakan bantuan *tools* yang tersedia di internet, hashkiller.co.uk



Gambar 2. *Screenshot* masukan hash:salt ke hashkiller.co.uk

Sumber :https://hashkiller.co.uk/Cracker/SHA1

Gambar 3. *Screenshot* hasil uji coba *cracking* dengan *hashkiller.co.uk*

Sumber : https://hashkiller.co.uk/Cracker/SHA1

Gambar 4. *Screenshot* hasil *cracking* dengan *hashkiller.co.uk*

Sumber : https://hashkiller.co.uk/Cracker/SHA1

Karena percobaan *cracking* dengan bantuan situs ini tidak membuahkan hasil, kita akan mencoba melakukannya secara manual dengan bantuan *python script*. *Plain text* dari *hash* yang diberikan terdiri atas 4 karakter bilangan desimal, dengan mengacu pada soal yang diberikan sebelumnya, "Hash breaker" yang dapat dilihat pada https://ringzer0team.com/challenges/56.

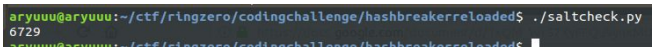
Berikut adalah *script python* yang digunakan untuk menguji kebenaran posisi penyisipan *salt* pada *plain text* untuk soal ini.

saltcheck.py

```
#!/usr/bin/python
from hashlib import sha1

expected =
"f9f61ae3eec17b94d3c1083b24e60a0142d7363d"
salt =
"57cb73e1fa5d0834310d57c4a04d03696c1243421
017c72d9fdca943a3236b0e"

temp = ""
i = 0
while(i <= 10000 and temp != expected):
    temp = sha1(str(i)+salt).hexdigest()
    i += 1

print i-1
```

Script tersebut akan mencoba melakukan hashing terhadap *string* yang terdiri atas 4 digit desimal dan membandingkannya dengan *message digest* yang didapat dari soal.

Ini adalah hasil yang didapat setelah menjalankan *script* ini.



Gambar 5. Hasil eksekusi *script* saltcheck.py

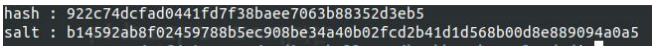
Karena hasil eksekusi merupakan *string* yang terdiri dari 4 karakter bilangan desimal, posisi penyisipan *salt* pada *plain text* benar.

Setelah mengetahui jenis *hash* yang digunakan dan posisi *salt* pada *plain text,* kita sudah siap untuk menyelesaikan soal ini. Karena *hash* dan *salt* yang diberikan berbeda untuk tiap sesi dan tiap sesi harus diselesaikan dalam waktu 3 detik, mengirimkan hasil yang kita dapat secara manual terasa mustahil, library *requests* akan sangat membantu dalam pengerjaan soal ini.

Langkah pertama dalam pengerjaan soal ini adalah, mendapatkan *hash* dan *salt* yang diberikan oleh soal. Kita dapat mendapatkan keduanya dengan mengirim *get request* ke halaman soal.

```
#!/usr/bin/python
from requests import get

def gethash(page):
    result = page[page.find("----- BEGIN
HASH -----<br />")+32:page.find("----- END
HASH -----<br />")-10]
    return result

def getsalt(page):
    result = page[page.find("----- BEGIN
SALT -----<br />")+32:page.find("----- END
SALT -----<br />")-10]
    return result

url =
"https://ringzer0ctf.com/challenges/57/"
sessid = {'PHPSESSID':[REDACTED]}

page = get(url, cookies=sessid)
thehash = gethash(page.text)
salt = getsalt(page.text)
print "hash : ", thehash
print "salt : ", salt
```

Gambar 6. Eksekusi *script* untuk mendapatkan hash dan salt



Gambar 7. Hasil Eksekusi *script*

Setelah *hash* dan *salt* didapat, langkah berikutnya adalah melakukan *brute force* untuk melakukan pencocokan *hash* dari *plain text* tebakan dengan *hash* yang diberikan.

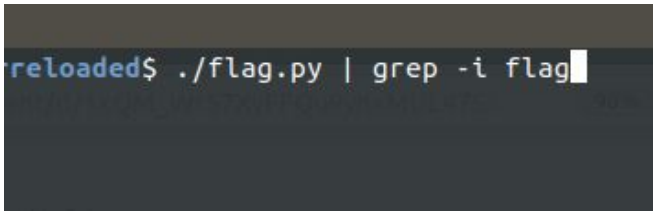
Terakhir, kita harus mengirim *get request* ke **https://ringzer0ctf.com/challenges/57/[clear_text]** . Berikut adalah *source code* yang digunakan untuk menyelesaikan soal tersebut.

flag.py

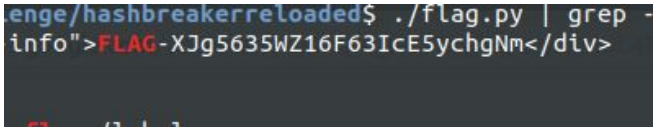
```
#!/usr/bin/python
from requests import get
from hashlib import sha1


def gethash(page):
    result = page[page.find("----- BEGIN
HASH -----<br />")+32:page.find("----- END
HASH -----<br />")-10]
    return result

def getsalt(page):
    result = page[page.find("----- BEGIN
SALT -----<br />")+32:page.find("----- END
SALT -----<br />")-10]
    return result

url =
"https://ringzer0ctf.com/challenges/57/"
sessid = {'PHPSESSID':'[REDACTED]'}

page = get(url, cookies=sessid)
thehash = gethash(page.text)
salt = getsalt(page.text)
print "hash : ", thehash
print "salt : ", salt
# print page.text


x = 0
ans = ""
while(ans != thehash):
    ans = sha1(str(x)+salt).hexdigest()
    x += 1


resp = get(url+str(x-1), cookies=sessid)
```

```
resp = resp.text
flag =
resp[resp.find("FLAG-"):resp.find("</div>"
)]

print resp
print flag
```

Berikut adalah hasil eksekusi *script* tersebut



Gambar 8. Eksekusi flag.py



Gambar 9. Hasil eksekusi flag.py

Setelah *script* selesai dijalankan, kita berhasil mendapatkan *flag* yang kita cari-cari.

Pemecahan *message digest* SHA-1 dengan *brute force* pada soal ini hanya mungkin dilakukan karena *string* asli yang dicari sudah diketahui formatnya, *string* relatif pendek dan karakter yang digunakan sangat sedikit sehingga pemecahan dapat berlangsung dengan sangat cepat.

IV. Kesimpulan

Hashing merupakan *one-way function* yang artinya *string* yang sudah ditransformasi menjadi *message digest* tidak dapat dikembalikan lagi menjadi *string* semula.

Satu-satunya pendekatan yang bisa digunakan untuk mendapatkan *string* asli dari *message digest hash* adalah dengan menggunakan pendekatan *brute force*, karena data dengan satu bit berbeda, ketika dilewatkan pada fungsi *hash* yang sama akan menghasilkan *message digest* yang berbeda sehingga kita harus memeriksa semua kemungkinan yang ada untuk memecahkannya.

String tebakan dilewatkan pada fungsi *hash* yang sama dengan *message digest* yang diberikan, kemudian dibandingkan dengan *message digest* tersebut.

Soal ini menjadi mungkin untuk dipecahkan karena *plain text* dari *message digest* terbatas hanya pada angka dan hanya terdiri atas 4 karakter, sehingga percobaan yang dibutuhkan untuk memecahkannya hanya 10^4 untuk mendapatkan seluruh kemungkinan *string*, jumlah yang terbilang sedikit untuk dikerjakan oleh komputer zaman sekarang.

Pada dunia nyata, dimana data yang dilewatkan pada fungsi *hash* tidak sesederhana itu yaitu dengan karakter yang begitu banyak ragamnya dan panjang yang tidak tentu, banyak percobaan yang harus dilakukan untuk memecahkan suatu *message digest hash* SHA1 dapat mencapai 2^{160}, jumlah yang luar biasa inilah yang membuat fungsi *hash* dipilih untuk mengamankan data-data penting seperti *password*.

Meskipun akan memakan waktu yang sangat lama, tetapi *brute force* masih menjadi satu-satunya pendekatan yang dapat memberikan hasil optimal untuk dalam persoalan ini.

V. Ucapan Terimakasih

Pertama-tama, saya ucapkan syukur kepada Allah SWT karena berkat rahmat dan karunia-Nya, saya diberi kesehatan dan kemampuan untuk menyelesaikan makalah ini dengan baik.Kemudian saya juga mengucapkan terima kasih sebesar-besarnya kepada tim dosen pengampu mata kuliah IF 2211 yaitu kepada Dr. Ir. Rinaldi Munir, MT., Dr. Masayu Leylia Khodra ST,MT, dan Dr. Nur Ulfa Maulidevi ST,M.Sc. yang telah membimbing dan menyampaikan materi terkait Strategi Algoritma. Semoga makalah ini dapat memberikan manfaat sebesar-besarnya kepada para pembaca.

References

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2017-2018/Fungsi-Hash-(2018).pdf

[2] http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2017-2018/SHA-(2018).pdf

[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Brute-Force-(2016).pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

M Algah Fattah Illahi 13517122